

A Novel LP-based Local Search Technique

– Fast and Quite Good –



TECHNISCHE
UNIVERSITÄT
DARMSTADT

Vom Fachbereich Informatik der
Technische Universität Darmstadt
zur Erlangung des akademischen Grades eines
Dr. rer. nat.

genehmigte Dissertation

von
Herrn Alaubek Avdil (M.Sc.)
aus der Ulaanbaatar, Mongolei

Referent: Professor Dr. Karsten Weihe
Korreferent: Professor Dr. Matthias Müller-Hannemann

Tag der Einreichung: 04. Juni 2009
Tag der mündlichen Prüfung: 17. Juli 2009

Darmstadt 2009

D 17

Acknowledgments

I want to express my deep and sincere gratitude to my supervisor Prof. Karsten Weihe for enabling my doctoral study, inspiring with scientific work and motivating and providing me pleasant research and work atmosphere. Without his trust and encouragement my research would not be done and this theses would not exist.

I am extremely grateful to Prof. Matthias Müller-Hannemann for his invaluable support and advice on my research, excellent and critical review to improve the presentation of my work, and for his friendship, and making it possible for my young family be close with me during my doctoral study in Germany.

I would like to express my warm and sincere thanks to the colleagues and members of Algorithmics Group at the Department of Computer Science of Technische Universität Darmstadt for their support and valuable comments on my research. My special acknowledgment goes to Dr. Roland Martin for his unforgettable support and encouragement during my doctoral study, and his detailed review and constructive criticism during the writing process of my dissertation.

I wish to extend my warmest thanks to all people, who have helped me with my work at the Department of Computer Science at Technische Universität Darmstadt.

My loving thanks to my dear wife, Raisa Kader, for her endless love and endurance, support and understanding of my work.

I am grateful to my parents and my brother who gave me a possibility to achieve my academic goal and for their love.

Abstract

We present and evaluate a specific way to generate good start solutions for local search. The start solution is computed from a certain LP, which is the modification of the underlying problem.

Generally speaking, we will look at the non-linear formulations of the problems and apply small modifications to transform the non-linear ingredients into linear ones. It is a requirement of our technique to work that the optimal basis solutions of the LP are feasible to the primary optimization problem. We consider four optimization problems: the directed Max-Cut problem with a source and a sink, and three variations of the Max- k -SAT problem with $k = 2$, $k = 3$ and $k = 4$. In each case, we define the modification such that the vertices of the LP are integral, and that the simplex method will not end up at infinity.

To compare our technique, we run local search repeatedly with random start solutions. Our technique produces consistently final solutions whose objective values are nearly identical to the best solutions from repeated random starts. The surprising degree of stability and uniformity of this result throughout all of our experiments on various classes of instances strongly suggests that we have consistently achieved nearly optimal solutions. Furthermore, an implementation of our technique to the Longest Directed Path problem with a source and a sink (in which we obtain an LP by incorporating flow-consistency inequalities) strongly supports our empirical findings. On the other hand, the run time of our technique is rather small, so the technique is very efficient and seemingly quite accurate.

Keywords:

Computations on discrete structures, Algorithms, Design, Experimentation, Polyhedral combinatorics, Max-Cut, Max-SAT, Max- k -SAT, Longest-Path.

Zusammenfassung

Wir präsentieren und evaluieren eine neuartige Methode, die gute Start-Lösungen für die lokale Suche generiert. Die Start-Lösung wird von einem bestimmten linearen Programm (LP) bestimmt, das eine Modifikation des zugrundeliegenden Problems ist.

Wir betrachten eine nicht-lineare Formulierung des Problems und wenden kleine Änderungen an, um die nicht-linearen Bestandteile der Formulierung in lineare Bestandteile umzuwandeln. Damit die Technik angewendet werden kann, ist es notwendig, dass die optimalen Basis-Lösungen des LPs zulässig für das primäre Optimierungsproblem sind. Wir untersuchen vier Optimierungsprobleme: das gerichtete Max-Cut Problem mit einem Start- und einem Endknoten, und drei Variationen des Max- k -SAT Problems mit $k = 2$, $k = 3$ und $k = 4$. In jedem Fall definieren wir die Modifikation so, dass die Eckpunkte des resultierenden LP ganzzahlig sind, und dass die Simplex-Methode nicht im Unendlichen endet.

Zum Vergleich mit unserer Technik benutzen wir eine wiederholte lokale Suche mit zufälligen Start-Lösungen. Unsere Technik produziert konsequent Lösungen, deren Zielwerte nicht allzu weit von den besten Lösungen aus wiederholten zufälligen Starts sind. Das überraschende Maß an Stabilität und der Einheitlichkeit der Ergebnisse in allen unseren Experimenten mit verschiedenen Klassen lassen den Schluss zu, dass wir konsequent nahezu optimale Lösungen erzielen. Darüber hinaus bestätigt eine Umsetzung unserer Technik auf das längste Pfad Problem zwischen zwei angegebenen Knoten (in denen wir das LP durch den Einbau der Fluss-Konsistenz-Ungleichungen herstellen) unsere empirische Befunde. Auf der anderen Seite ist die Laufzeit unserer Technik sehr klein, so dass diese sehr effizient ist.

Schlagwörter:

Berechnungen auf diskreten Strukturen, Algorithmen, Design, Experimentieren, Polyedrische Kombinatorik, Max-Cut, Max-SAT, Max- k -SAT, Längster-Pfad.

Contents

Acknowledgments	III
1. Introduction	1
1.1. Formulation of the Problems	3
2. Integer Linear Programming	9
2.1. Introduction	9
2.2. Computational Complexity	11
2.3. Relaxation and Valid Inequalities	11
2.3.1. Linear programming relaxation	12
2.3.2. Convexity Cuts	15
2.3.3. Lift and Project	17
2.3.4. Lagrangian relaxation	23
2.4. Cutting plane algorithms	25
2.4.1. Cutting plane algorithms for 0-1 IPs	29
2.5. Branch and Bound Algorithms	34
2.6. Branch and Cut Algorithms	37
3. State of the Art	41
3.1. Algorithmic Results for the Max-Cut and Max-SAT	41
3.1.1. Non-approximability Results	43
3.2. Algorithmic Results for the Longest Path Problem	45
3.3. Theoretical Bound for the Max-2-SAT	47
3.4. Heuristic Methods	48
3.5. Hybrid LP-based Approaches	57

4. Local Search Starting From an LP Solution	61
4.1. Introduction	61
4.2. Directed Max-Cut with Source and Sink	62
4.3. Max- k -SAT	64
4.4. The Longest Path with Source and Sink	65
5. Computational Study	69
5.1. Experimental Setup	69
5.2. Directed Maximum Cut with Source and Sink	71
5.2.1. Generation of Test Instances	71
5.2.2. Experimental Evaluation	74
5.3. Maximum Satisfiability	84
5.3.1. Generation of Instances	84
5.3.2. Max-2-SAT	85
5.3.3. Max-3-SAT	97
5.3.4. Max-4-SAT	102
5.4. The Longest Path with Source and Sink	106
5.4.1. Generation of Instances	106
5.4.2. Summary of Computational Results	108
6. Conclusion	117
List of Tables	121
List of Figures	123
Bibliography	127
A. Types of Graphs Generated by rudy	139
B. Some Details of Computational Results	141
B.1. Max-Di-Cut: 100 individual runs of TV	141
B.2. Max-2-Sat: Computational Results for the 2nd class	141
B.3. Max-2-Sat: 60 individual runs of the TV	147

Chapter 1

Introduction

The existence of multiple locally optimal solutions in combinatorial optimization problem makes it difficult to solve them. Therefore, the local search might end up in the local optimal solution, without reaching a global optimum. Many efficient heuristic methods are based on local search. When the local search reaches local optima or plateaus, these methods use techniques that help the search to escape from the local optima or from the plateaus. Some techniques lead to the continuation of the search from a random point, or to the continuation of the search from some constructed points, or they avoid to visit some of the points with attributes that are found and learned in the search history. Even though these strategies are called and defined differently in combinatorial optimization, the search strategies can be categorized into several classes: multi-start, memory based, variable neighborhood, population based and randomized. Some heuristics use only one kind of strategy, while others combine two or several strategies.

Clearly, the quality of the primitive local search algorithm depends crucially on the choice of the start solution. In this thesis, we will consider a linear programming (LP) based technique that evidently generates start solutions of an apparent high quality. This technique is developed by Avdil and Weihe [10]. Roughly speaking, we will look at a non-linear formulation of the problem and apply small modifications to transform the non-linear ingredients into linear ones. In each case, we define the modification such that the vertices of the resulting LP are integral, and that the simplex method will not end up at infinity (although the polyhedron itself will be unbounded in general).

This technique might apply to a variety of optimization problems but the adaptation to a given problem does not seem to be entirely trivial. To apply our method to a given problem,

an LP has to be found such that the optimal basis solutions to the LP are actually feasible solutions to the optimization problem. Therefore, the existence of such LP is required for the application of our method. We use the directed MAX-CUT problem (MAX-DI-CUT) with source and sink and three variations of the MAX- k -SAT problem with $k = 2$, $k = 3$ and $k = 4$ as examples. Furthermore, we implement our technique on the LONGEST DIRECTED PATH problem with source and sink through an alternative way, in which some of the constraint inequalities of LP are originated from an another optimization problem.

A two-phase algorithm resulting from our LP-based local search technique can be seen as a hybrid algorithm that incorporates the LP and local search.

In a computational study we tested the behavior and the stability of our technique on various classes of test instances. For our experiments we used, beside self written graph generators, a public domain and machine independent graph generator `rudyl`¹, written by G. Rinaldi. Some of the test instances have been created by Helmberg and Rendl [91], and were used to test their algorithm implementing a bundle method for solving semidefinite programming (SDP). For the MAX-2-SAT problem we additionally used a graph generator written by Jagota and Sanchis [95]. This generator generates graphs with known sizes of cliques (actually designed for the max-clique problem). For the MAX-3-SAT and MAX-4-SAT problems, we additionally used a random CNF-formula generator written by B. Selman (personal communication, [142]). The random graph generator `GTgraph` [11] generated some of the MAX-4-SAT instances and the LONGEST DIRECTED PATH instances.

We tested our technique against a *reference technique*: a repeated local search from random start solutions. This technique has turned out to be particularly appropriate for comparisons since – evidently – it consistently produces near-optimal solutions. Although the reference technique is sufficiently different, it does turn out that – without even one exception – both algorithms produce solutions whose objective values do not differ from each other by more than a small percentage continuously throughout all test instances. Yet, for the overwhelming majority of all cases, the difference is truly marginal. For the MAX-2-SAT problem, the asymptotic bounds of expected optimal value are available for a certain type of instances. These bounds are very tight to our technique’s results.

How can, then, such a strong consistent coincidence of empirical results of two (resp. three in the case of MAX-2-SAT) sufficiently different methods be explained? In our opinion, the only plausible explanation is that both (resp. three) approaches produce – without even

¹<http://www-user.tu-chemnitz.de/helmberg/semidef.html>

one exception – nearly optimal solutions.

The adaptation of our technique to the LONGEST DIRECTED PATH problem with source and sink, and the comparison to the reference technique confirms that our strong empirical results of the coincidence of different approaches could not have happened by a chance. In the empirical study for the LONGEST DIRECTED PATH, there are a number of instances where our technique is significantly better than reference technique, and there are some instances where reference technique is noticeably better than our technique. Even if in average, our algorithm found better solutions than reference technique, we could not conclude that our technique found near optimal solutions to this problem.

This thesis is organized as follows. In the next section, we formulate the problems MAX-DI-CUT, the MAX- k -SAT, where $k \geq 2$, and the LONGEST DIRECTED PATH. Chapter 2 is devoted to the common algorithmic methods for solving the integer program, including relaxation methods and cutting plane algorithms for binary optimization. In Chapter 3, we review the algorithmic results and the heuristic methods which are implemented and applied to the maximum cut, maximum satisfiability and longest path problems. In this chapter we also review the hybrid LP-based approaches proposed for combinatorial optimization problems. The core of this thesis is Chapter 4 and Chapter 5. We introduce our technique for the MAX-DI-CUT and the MAX- k -SAT in Chapter 4. In this chapter we also present the adaptation of our technique to the LONGEST DIRECTED PATH problem. Finally, we present the experimental study in Chapter 5 in more detail. We conclude with Chapter 6.

1.1. Formulation of the Problems

In this section we first formulate the maximum cut, the maximum satisfiability and the longest path problems, and their variations. Then we give some facts considering these problems.

We consider graphs without loops or multiple edges. Henceforth, the word *edge* is reserved for undirected graphs, and the word *arc* is for directed graphs. Given an undirected graph $G = (V, E)$ with node set V and edge set E we use the notation $n := |V|$ and $m := |E|$ for the order and size of G , respectively. We shall assume the node set is denoted by $V = \{1, \dots, n\}$ and edge $e \in E$ connecting node v and w is denoted by $e = (v, w)$. For any node $v \in V$ let N_v denote the set of its neighbors, $N_v = \{w \in V : \exists (v, w) \in E\}$. For a node v we denote the set of edges incident with v by δ_v and let d_v denote the *degree* of v ,

$d(v) = |\delta_v|$. Analogously, for the given directed graph $G = (V, A)$ with node set V and arc set A , we note the size and order of G as $n := |V|$ and $m := |A|$. The arc $a \in A$ pointing from v to w is denoted by $a = (v, w)$. We say that $a = (v, w)$ *leaves* v and *enters* w . For any arc (v, w) , v is called an *inneighbor* of w and w is called an *outneighbor* of v . For a node $v \in V$, N_v^{out} denotes the set of *outneighbors* of v and N_v^{in} denotes the set of *inneighbors* of v . For any node $v \in V$ we denote the set of arcs leaving v by δ_v^{out} , and the set of arcs entering v by δ_v^{in} :

$$\delta_v^{out} = \{a \in A : a = (v, w), w \in V\} \text{ and } \delta_v^{in} = \{a \in A : a = (w, v), w \in V\}.$$

For each node $v \in V$, d_v^{out} denotes the *outdegree* and d_v^{in} denotes the *indegree*, $d_v^{out} = |\delta_v^{out}|$ and $d_v^{in} = |\delta_v^{in}|$, respectively. A node of indegree 0 is called a *source* and a node of outdegree 0 is called a *sink*.

Maximum Cut Problem. Assume that for the undirected graph $G = (V, E)$ we are given an edge-weight function $c : E \rightarrow \mathbb{R}_0^+$.

A *cut* is a partition of node set V into two subsets, $S \subset V$ and $\bar{S} = V \setminus S$, and the weight of a cut – $C(S)$ – is defined as the sum of edge-weights that have one end in S and the other in \bar{S} , i.e.,

$$C(S) := \sum_{\substack{e=(v,w) \\ v \in S, w \notin S}} c_e.$$

Similarly, the weight of a cut in the directed graph is defined as the sum of arc-weights that point from S to \bar{S} :

$$C(S) := \sum_{\substack{a=(v,w) \\ v \in S, w \notin S}} c_a.$$

The maximum cut problem can be formulated as:

MAX-CUT PROBLEM

Instance: Graph G and weight function c .

Objective: Maximize the cut-weight.

A restricted version of the problem, where all edges have uniform weights, is called the SIMPLE MAX-CUT problem. In the directed graphs we call the problem DIRECTED MAX-CUT

problem (or MAX-DI-CUT). Let us denote the weight of maximum cut:

$$mc(G, c) = \max_{S \subset V} C(S).$$

Assume that we are given a directed graph $G = (V, A)$, a nonnegative weight function $c : A \rightarrow \mathbb{R}_0^+$ and two nodes s and t (a source and a sink). An (s, t) -cut is a partition of V into two subsets, S and T , such that $s \in S$ and $t \in T$. The weight of the (s, t) -cut is defined similarly to the cut-weight in the directed graph, as the sum of the arc-weights that point from S to T :

$$C(S, T) := \sum_{\substack{a=(v,w) \\ v \in S, w \in T}} c_a.$$

In this thesis we focus on the following problem:

DIRECTED MAX-CUT PROBLEM WITH SOURCE AND SINK

Instance: Directed graph G , nonnegative weight function c and two nodes – s and t .

Objective: Maximize the weight of (s, t) -cut.

The problem formulations, which find the cut with maximum weight and find only the value of maximum cut are different. But in practice, knowing only the value of the maximum cut and not the maximum cut itself is not worthy.

The simple lower bounds on the maximum cut are given by:

Proposition 1.1.1 *For every graph $G = (V, E)$ and weight function c , $mc(G, c) \geq \frac{1}{2} \sum_{e \in E} c_e$.*

Proof: Let S be a local optimum, i.e., adding to S or moving out from S any node does not increase the total weight of the cut, and denote $\bar{S} = V \setminus S$. Then for each $v \in S$ the total weight of edges connecting v with nodes of \bar{S} is at least as heavy as total weight of edges connecting v with other nodes of S , otherwise, moving v into \bar{S} yields better solution. The analogous statement holds for each $v \in \bar{S}$. Therefore, $\sum_{e \in E} c_e - C(S) \leq C(S)$. \square

Proposition 1.1.2 *For every directed graph $G = (V, A)$ and weight function c , $mc(G, c) \geq \frac{1}{4} \sum_{a \in A} c_a$.*

Proof: Assume graph $G' = (V, A')$ created in a way that for each arc $a \in A$ pointing from v to w with weight c_a , a new opposite oriented arc $a' = (w, v)$ with weight $c_{a'} = c_a$ is drawn. Then the maximum cut in G' is $mc(G', c) \geq \frac{1}{2} \sum_{a \in A'} c_a$. The original arcs (of A) across the cut have $\frac{1}{2}$ of the weight. \square

Maximum Satisfiability Problem. Assume conjunctive normal form (CNF) formula F on a set of binary variables X . We denote $n := |X|$ and $m := |F|$ as number of variables and clauses, respectively. Each clause is a disjunction of literals, where literals are $x \in X$ or its negation \bar{x} . Moreover, we denote $X = \{x_1, \dots, x_n\}$, the negations of variables $\bar{X} = \{\bar{x}_1, \dots, \bar{x}_n\}$ and $F = \{C_1, \dots, C_m\}$. A truth assignment $T : X \rightarrow \{0, 1\}$ is a mapping which assigns the value 0 (*false*) or 1 (*true*) to each variable x_i , $i \in \{1, \dots, n\}$. Then the maximum satisfiability problem can be formulated:

MAX-SAT PROBLEM

Instance: CNF-formula F on set of variables X .

Objective: Find a truth-assignment T that satisfies the maximum number of clauses in F .

In this thesis, we consider some restricted versions of the MAX-SAT problem. If we are given a CNF-formula F where the clauses contain at most $k \in \mathbb{Z}^+$, $k \geq 2$, literals, then the problem is called MAX- k -SAT. Additionally, if every clause consists of exactly k literals, $k \geq 2$, then the problem variant is called MAX-E k -SAT. There are also weighted versions of these variants of the maximum satisfiability problem. Assume that a weight $w_C \in \mathbb{R}$ is associated to each clause C of F . We can then define a weight of truth-assignment T as the sum of clause-weights that are satisfied by T :

$$W(T) := \sum_{\substack{C \in F: \\ T \text{ satisfies } C}} w_C.$$

Then the weighted MAX-SAT, the weighted MAX- k -SAT and the weighted MAX-E k -SAT problems ask to find a truth-assignment with maximum weight.

The MAX-CUT and MAX-3-SAT problems are two of the problems that have been shown to be \mathcal{NP} -complete in Karp's famous paper [98].

Theorem 1.1.3 [34, 65, 98] *The following problems are \mathcal{NP} -Hard*

- ★ MAX-CUT
- ★ SIMPLE MAX-CUT
- ★ MAX-SAT *and*
- ★ MAX- k -SAT.

Longest Path Problem. Given a graph (directed or undirected) $G = (V, E)$ with n nodes and m edges. A *simple path* is the sequence of distinct vertices v_1, \dots, v_k such that $v_i v_{i+1} \in E$, $1 \leq i \leq k-1$. Hereafter, we will consider only simple paths, therefore, we refer to “simple path” just as “path”. A length of the path is the sum of all edges on the path. Then the longest path problem can be formulated:

LONGEST PATH PROBLEM

Instance: A Graph $G = (V, E)$.

Objective: Find a path with maximum length.

In this thesis we consider a restricted version of the longest path problem, in which the objective is to find the longest path between two given nodes in a directed and weighted graph. An input consists of a directed graph $G = (V, A)$, a positive cost (or length) $c_a \in \mathbb{R}^+$ on each arc $a \in A$, and two nodes, s and $t \in V$ (source and sink). Recall that the source has no entering arc and the sink has no leaving arc, $\delta_s^{in} = \emptyset$ and $\delta_t^{out} = \emptyset$. An $s-t$ path is a sequence of distinct vertices $P = \{v_1, v_2, \dots, v_k\}$ such that for any $1 \leq i \leq k-1$, (v_i, v_{i+1}) is an arc, $(v_i, v_{i+1}) \in A$, and $v_1 = s$ and $v_k = t$. The length of the $s-t$ path P is the sum of lengths of arcs which are on the path

$$C(P) := \sum_{i=1}^{k-1} c_{v_i v_{i+1}}.$$

We define this restricted version of the longest path problem as:

LONGEST DIRECTED PATH PROBLEM WITH SOURCE AND SINK

Instance: A directed graph $G = (V, A)$, a positive length function c on the arcs and two nodes s and t .

Objective: Find an $s-t$ path with maximum length.

Sometimes, we write “LONGEST DIRECTED PATH” for the above problem where the path is clearly meant to be between source and sink.

Chapter 2

Integer Linear Programming

2.1. Introduction

In the field of optimization problems, the decisions and solutions in the real world are mostly discrete, such as the quantity of items or choosing the options from finite set of alternatives. An optimization problem, in which the variables must take integer values, is called Integer Program and the subject of solving such programs is called integer programming (IP). The problems in which some variables are restricted to take integer values and some variables that can take fractional values are called mixed integer programs (MIPs). Generally, integer programs consider the nonlinear objective and constraint functions, but these are subject of another research discipline, integer nonlinear programming. IP can be formulated as follows:

$$cx \rightarrow \max, \quad x \in X \tag{2.1}$$

where $X = \{x \in \mathbb{Z}_+^n : Ax \leq b\}$, $A \in \mathbb{Z}^{m \times n}$ is an integer $m \times n$ matrix, $b \in \mathbb{Z}^m$ is an m -vector, and $c \in \mathbb{Z}^n$ is an n -vector.

There are many real world IPs where the variables can take only one of two values (such as yes/no, in/out) and can be modeled as 0 and 1. The IPs, in which the variables can take only 0's and 1's are called 0-1 integer programs, or binary optimization problems. For example we consider the MAX-DI-CUT problem with source and sink. Assume that given an instance of MAX-DI-CUT with source and sink {di-graph $G = (V, A)$, cost function $c : A \rightarrow \mathbb{R}_+$, $s, t \in V$ }. An (s, t) -cut may be identified with its characteristic vector $x = (x_1, \dots, x_n) \in \{0, 1\}^n$, $n = |V|$: $x_v = 1$ if, and only if, $v \in S$. For each arc $a = (v, w) \in A$, we define a variable $y_a \in \{0, 1\}$ such that $y_a = 1$ if, and only if, $v \in S$ and $w \in T$. The

problem objective can be written

$$\sum_{a \in A} c_a \cdot y_a \longrightarrow \max.$$

The constraints are

$$\begin{aligned} y_a &\leq 1 - x_w & \forall a = (v, w) \in A \\ y_a &\leq x_v & \forall a = (v, w) \in A \\ x_s - x_t &\geq 1 \\ x_v &\in \{0, 1\} & \forall v \in V \\ y_a &\in \{0, 1\} & \forall a \in A \end{aligned}$$

The constraints $y_a \leq 1 - x_w$ and $y_a \leq x_v$ allow the variable y_a to take value “1” only for the arcs $a = (v, w) \in A$ leaving s -side and entering t -side, and force the variable y_a to become “0” for all other arcs.

Another example of 0-1 IP is the unconstrained facility location problem (UFL). We are given a set of locations F , where the facilities can be built, facility building cost at the i -th location f_i , $i = 1, \dots, n$, $n = |F|$, and a set of clients D , that should be supplied from the facilities. Let $m = |D|$. Moreover the shipping cost for unit product (demand) from i -th location to j -th client is $c_{ij} \geq 0$, $i = 1, \dots, n$, and $j = 1, \dots, m$. Assume that each location can supply the demand of all clients and each client has a unit demand. The objective is to find the locations where facilities will be built and to assign each client to some location. We can model the UFL as following: let x_i , $i = 1, \dots, n$, indicates the location is whether built or not, i.e., $\forall i \in \{1, \dots, n\}$ if $x_i = 1$ then the facility will be built on i -th location, otherwise if $x_i = 0$ then the facility will not be built. Let the variables y_{ij} , $i = 1, \dots, n$ and $j = 1, \dots, m$, imply the supply-assignment, where $y_{ij} = 1$ means that j -th client is supplied by i -th location. The problem objective can be written

$$\sum_{i=1}^n f_i x_i + \sum_{\substack{1 \leq i \leq n \\ 1 \leq j \leq m}} c_{ij} y_{ij} \longrightarrow \min.$$

The constraints are formulated as

$$y_{ij} \leq x_i \quad \forall i \in \{1, \dots, n\}, \forall j \in \{1, \dots, m\}$$

$$\begin{aligned}
\sum_{i=1}^n y_{ij} &\geq 1 & \forall j \in \{1, \dots, m\} \\
x_i &\in \{0, 1\} & \forall i \in \{1, \dots, n\} \\
y_{ij} &\in \{0, 1\} & \forall i \in \{1, \dots, n\} \\
& & \forall j \in \{1, \dots, m\}.
\end{aligned}$$

The constraints $y_{ij} \leq x_i$, $\forall j \in \{1, \dots, m\}$ indicate that no client can be supplied from the i -th location unless the facility is built there.

2.2. Computational Complexity

According to Karp [98], the decision version of integer program was proved to be \mathcal{NP} -Complete. The decision version of IP can be formulated as following.

IP-DECISION

Instance: An integer $m \times n$ matrix $A \in \mathbb{Z}^{m \times n}$, m -vector $b \in \mathbb{Z}^m$, a nonnegative integer n -vector $c \in \mathbb{Z}_+^n$ and an integer $B \in \mathbb{Z}$.

Task: Is there a vector $x \in \mathbb{Z}^n$ such that $Ax \geq b$ and $cx \geq B$?

IP is \mathcal{NP} -hard, so is 0-1 integer programming. Even some restricted versions of the IP remain to be \mathcal{NP} -hard. The following results summarize the intractability of some restricted versions of IP.

Theorem 2.2.1 [98] *The IP-DECISION is \mathcal{NP} -complete.*

Analogously, the decision version of 0-1 IP is hard to solve.

Proposition 2.2.2 *The decision version of 0-1 integer programming is \mathcal{NP} -complete.*

2.3. Relaxation and Valid Inequalities

In this section we talk about the representation of integer program (2.1) by linear program. We consider the IP formulated as

$$cx \rightarrow \max, \quad x \in X$$

where $X = \{x \in \mathbb{Z}_+^n : Ax \leq b\}$.

2.3.1. Linear programming relaxation

An LP relaxation is obtained from IP by dropping the integrality constraints:

$$cx \rightarrow \max, \quad x \in P$$

where $P = \{x \in \mathbb{R}_+^n : Ax \leq b\}$. The convincing character of the LP relaxation is that it delivers an upper bound for the optimal solution value. We can formulate an infinite number of LP relaxations of IP, and by relaxation we mean the relaxed polyhedron which contains the original feasible set. Therefore, if we find a tighter polyhedron, then we could find tighter upper bound for an objective functions value. As we know, there have been many ways and methods developed to construct specifically the tight polyhedrons. The polyhedral theory of IP has been widely studied [161, 113].

If the optimal solution of LP relaxation is integral, then it is optimal to IP. Otherwise, we want to find an LP, of which the optimal solution is integral. By definition a polyhedron P is integral if every face contains an integer point. By the integer Farkas lemma [138] this is equivalent to the fact that every supporting hyperplane of P contains an integer point. The main idea here is to look at every supporting hyperplane, and shift it closer to the convex hull of X until it contains an integer point.

The inequality $ax \leq b$ is *valid* for polyhedron P if $ax \leq b, \forall x \in P$. If the inequality $ax \leq b$ is valid for polyhedron P and in addition if $\exists x \in P$ such that $ax = b$, then $ax \leq b$ is a *supporting inequality* and $\{x \in \mathbb{R}^n : ax = b\}$ is a supporting hyperplane. If the inequality $ax \leq b$ is not a supporting inequality of polyhedron P , but $\exists x \in P$ such that $ax \leq b$ and $\exists y \in P$ such that $ay > b$, then the hyperplane $ax = b$ cuts the polyhedron P , and is called a *cutting plane*. For the supporting inequality $ax \leq b$ of P , a subset $F = \{x \in P : ax = b\}$ of P is called a *face*. A *facet* of P is an inclusionwise maximal face F of P with $F \neq P$.

Let $\pi x \leq \pi_0$ be a supporting inequality of P with π integer. $P \subseteq \{x \in \mathbb{R}^n : \pi x \leq \pi_0\}$. Let Φ denote a set of all supporting inequalities of P with integral left-hand-side coefficients and

$$Q^1 := \bigcap_{(\pi, \pi_0) \in \Phi} \{x \in \mathbb{R}^n : \pi x \leq \lfloor \pi_0 \rfloor\}. \quad (2.2)$$

Then $\text{conv}(X) \subseteq Q^1$. Let us apply the same procedure on Q^1 , and continue iteratively

$$Q^0 := P \text{ and } Q^{t+1} := (Q^t)^1.$$

Then we have

$$P = Q^0 \supseteq Q^1 \supseteq \dots \supseteq \text{conv}(X).$$

Chvátal [33] showed that if P is polytope, then $\text{conv}(X)$ can be obtained after finite number of these iterations. Schrijver [137] showed the same result for an arbitrary rational polyhedron.

Theorem 2.3.1 [33, 137] *Let P be a rational polyhedron. Then*

- I) Q^1 is a polyhedron;
- II) There exists a finite number t such that $Q^t = \text{conv}(X)$.

We describe how to generate linear programming relaxations (tighter polyhedrons). Denote $N = \{1, \dots, n\}$. Let x^* be an optimal solution of the LP relaxation and $B \subseteq N$ be a basis of A with $x_B^* = A_B^{-1}b - A_B^{-1}A_D x_D^*$ and $x_D^* = 0$, where $D = N \setminus B$. If x^* is integral, then it is also optimal to $\text{conv}(X)$. Otherwise, at least one of the values x_B^* is fractional; moreover, let $i \in B$ be the index of fractional component of x^* . Every feasible integral solution $x \in X$ satisfies $x_B = A_B^{-1}b - A_B^{-1}A_D x_D$, therefore

$$(A^{-1})_i b - \sum_{j \in D} (A^{-1})_i A^j x_j \in \mathbb{Z}. \quad (2.3)$$

By denoting the fractional part of $a \in \mathbb{R}$ as $f(a)$, $f(a) = a - \lfloor a \rfloor$, we restate

$$\lfloor A_{R^i}^{-1} b \rfloor + f(A_{R^i}^{-1} b) - \sum_{j \in D} (\lfloor A_{R^i}^{-1} A_{C^j} \rfloor + f(A_{R^i}^{-1} A_{C^j})) x_j \in \mathbb{Z}.$$

Moreover, after subtracting integer and adding integer multiples of x_j , it remains integer

$$f(A_{R^i}^{-1} b) - \sum_{j \in D} (f(A_{R^i}^{-1} A_{C^j})) x_j \in \mathbb{Z}. \quad (2.4)$$

Since $0 \leq f(a) < 1$ for $\forall a \in \mathbb{R}$,

$$f(A_{R^i}^{-1} b) \leq \sum_{j \in D} (f(A_{R^i}^{-1} A_{C^j})) x_j, \quad (2.5)$$

is valid for $\text{conv}(P)$. On the other hand, it is violated by the current LP relaxation solution x^* , since $x_D^* = 0$ and $f(A_{R^i}^{-1} b) = f(x_i^*) > 0$. After subtracting $x_i + \sum_{j \in D} A_{R^i}^{-1} A_{C^j} x_j = A_{R^i}^{-1} b$

from (2.5) we obtain

$$x_i + \sum_{j \in D} \lfloor A_{Ri}^{-1} A_{Cj} \rfloor x_j \leq \lfloor A_{Ri}^{-1} b \rfloor, \quad (2.6)$$

which, when right-hand-side is not rounded, is a supporting inequality with integral left-hand-side, therefore a member of Φ . Adding this inequality to the constraint system $Ax \leq b$ keeps the principle that all data are integral. Thus, the slack variable that is to be introduced for the new inequality can be required to be integral as well and the whole procedure can be iterated. In fact, Gomory's cutting plane method [81, 80] for integer linear programming adds this inequality to the constraint system and iterates a whole procedure. Gomory proved that alternately applying simplex method and adding cutting planes leads to a finite algorithm, that means, after adding a finite number of inequalities an integer optimal solution is found. (We will discuss the cutting plane approaches later in Section 2.4.) This indicates that if $cx \leq c_o$ defines a facet of $\text{conv}(X)$, and Gomory's cutting plane method is applied to the IP $\max\{cx : Ax = b, x \in \mathbb{Z}_+^n\}$, the inequality $cx \leq c_o$ lies in Q^t for some finite t . Therefore, Gomory's algorithm gives a proof for Theorem 2.3.1.

There are several simple characterizations of valid inequalities. The first is any nonnegative linear combinations of the valid inequalities for K are valid for K .

Rounding

If $ax \leq b$ is valid for X , where a is integral, then $ax \leq \lfloor b \rfloor$ is valid for X .

Disjunctive inequalities

We combine two inequalities, where each of them is valid for the partition of X , in order to obtain a valid inequality for X . Let X^1 and X^2 be the partition of $X = X^1 \cup X^2$, and $a^1x \leq b^1$ is valid for X^1 and $a^2x \leq b^2$ is valid for X^2 . Then

$$\sum_{i=1}^n \min(a_i^1, a_i^2) x_i \leq \max(b^1, b^2)$$

is valid for X .

Superadditive inequalities

A function $f : D \rightarrow \mathbb{R}$, $0 \in D$, $f(0) = 0$ is called *superadditive over D* if

$$f(d_1 + d_2) \geq f(d_1) + f(d_2) \quad \forall d_1, d_2 \in D,$$

and nondecreasing over D if

$$d_1, d_2 \in D, d_1 < d_2 \Rightarrow f(d_1) \leq f(d_2).$$

Proposition 2.3.2 *If f is superadditive and nondecreasing over \mathbb{R}^m then,*

$$\sum_{i=1}^n f(a^i)x_i \leq f(b)$$

is a valid inequality for $P = \{x \in \mathbb{Z}_+^n : Ax \leq b\}$.

The above given three methods for generating valid inequalities are actually robust as the following results will show.

Theorem 2.3.3 [112] *If $\pi x \leq \pi_0$ is a valid inequality for $P = \{x \in \mathbb{Z}_+^n : Ax \leq b\} \neq \emptyset$, then:*

1. *There exists superadditive, nondecreasing function f such that $f(a^j) \geq \pi_j$, $j = 1, \dots, n$ and $f(b) \leq \pi_0$.*
2. *The inequality $\pi x \leq \pi_0$ or a valid inequality that dominates it can be generated by starting with the inequalities $Ax \leq b$ and $x \geq 0$, and then taking linear combinations and rounding a finite number of times.*
3. *If $x \in \{0, 1\}^n$ the inequality $\pi x \leq \pi_0$ or a valid inequality that dominates it can be generated by starting with the inequalities $Ax \leq b$ and $0 \leq x \leq 1$ and then taking linear combinations and rounding a finite number of times.*

From the above results we can see that a finite algorithm can be constructed by applying these procedures iteratively, which enables the building of tighter LP relaxations. However, these steps could rise exponentially large, so it might be not practical. Still, this idea can be applicable on some IPs, which have known valid inequalities, that define facets of $\text{conv}(X)$.

2.3.2. Convexity Cuts

The ideas of convexity cut were first introduced in the context of concave programming by Tuy [153], and in the context of integer programming by Balas [12] and Young [164] (intersection cut). Glover [69] extended the original integer programming development to cover the general convex sets. To describe the convexity cuts we follow the specifications of Glover and Laguna [75].

The primary idea is to identify a convex set that contains the linear programming solution (fractional) in its interior but no integer feasible solutions. Then extending the edges of the polyhedral cone associated with LP solution until it intersects with a convex set or until it becomes computationally expensive to determine the intersection. Then the cut is determined by passing a hyperplane through the extreme endpoints of these extended edges. Let x^0 denote the basic extreme point solution obtained by simplex method solving the corresponding LP. The point x^0 corresponds to the vertex of polyhedral cone associated with LP. An extension of the edge from x^0 corresponds to an assignment of positive values to a selected nonbasic variable, holding each of the other variables at zero. Let D denote the set of current nonbasic variables and B denote the set of current basic variables. Let I denote the set of indices of the integer variables. To be easily understood we assume that, each variable x_i , $1 \leq i \leq n$ satisfies the bounds $U_i \geq x_i \geq 0$, where U_i may be infinity. A polyhedral LP cone of which x^0 is the vertex is a region spanned by the edges

$$x^h = x^0 - S_h u_h, \quad \text{for } u_h \geq 0, h \in D,$$

where S_h is the current tableau vector associated with the nonbasic variable x_h , and u_h is the parameter identifying the change in the value of x_h from its lower and upper bound value which it receives at x_0 point. Let s_{hi} denote the entries of S_h . For all nonbasic variables the entries of S_h are zero, except for x_h , $s_{hi} = 0$, $\forall i \in D$, $i \neq h$, which has a coefficient 1 or -1 . We choose the sign usage for S_h that yields a coefficient for x_h of $s_{hh} = 1$ if x_h is currently at its lower bound at point x^0 , and of $s_{hh} = -1$ if x_h is currently at its upper bound at point x^0 . By this sign usage if x^0 is a feasible extreme point of the LP, then the feasible extreme points adjacent to x^0 are points x^h that occur for nonnegative values of u_h , and for strictly positive values except under degeneracy. We assume that the components of x^0 that associate with the integer variables x_i , $i \in I$, have non-integer values. The construction procedure of convexity cuts described as follows.

Convexity Cut Construction

- Step 1. Identify a closed convex region whose interior includes x^0 but no feasible integer solutions.
- Step 2. Extend each edge of the LP polyhedral cone until it meets the boundary of the convex set.
- Step 3. Pass a hyperplane through the endpoints of the edges of the LP basis cone

where they intersect the boundary of the convex set. Letting u_h^* identify the value of u_h that corresponds to the point of intersection for edge h , the hyperplane can be expressed as the set of points

$$\sum_{h \in D} \left(\frac{1}{u_h^*} \right) u_h = 1$$

where $\frac{1}{u_h^*} = 0$ if u_h^* is infinity. This can be expressed in terms of the nonbasic variables x_h by substitution using the identity $u_h = x_h$ or $u_h = U_h - x_h$, according to whether x_h is nonbasic at its lower or upper bound in the current LP solution x^0 .

A simple example of a convex region can be given by the polyhedron $v \leq x_i \leq v + 1$, where $x_i, i \in I$.

The construction procedure creates two half spaces associated with the hyperplane. One can replace “=” by “ \leq ” in the defining equation and includes all points that lie on the side of hyperplane that contains the LP solution. The other replacement is “=” by “ \geq ” in the defining equation and includes all points that lie on the other side of the hyperplane, hence cuts off the LP vertex, which assigns x_j a fractional value.

Theorem 2.3.4 [75] *The half space*

$$\sum_{h \in NB} \left(\frac{1}{u_h^*} \right) u_h \geq 1$$

that excludes the LP vertex contains all the feasible integer solutions, and the associated hyperplane is a valid cut.

Glover and Laguna [75] proposed a heuristic method – *cut search method* – for the mixed integer programs, which uses the convexity cuts. Cut search process makes it possible to solve a simple restricted mixed integer program to obtain the best solution from a given collection of implicit candidate solutions, and to simultaneously generate a cutting plane by reference to this collection. For more details the reader is referred to [75].

2.3.3. Lift and Project

For 0-1 mixed integer programs there are methods proposed, which are called *lift-and-project*, an another way of strengthening the linear programming relaxation. The idea

is to reformulate the problem into a higher dimensional space, where a more convenient formulation may give a tighter relaxation. One then has a choice between working with this tighter relaxation in the higher dimensional space, or restating it back onto the original space. In the latter case, the procedure can be viewed as a method for generating valid inequalities (cutting planes) in the original space. Reformulating the problem into higher space is called *lifting*, and restating back into the original space is called *projecting*, therefore the method is called lift-and-project. The versions of this approach differ in how the lifting and projection are performed [16, 104, 147]. We explain the ideas of methods proposed by Balas, Ceria and Cornuéjols [16] in more detail and show the connection to others.

Assume 0-1 mixed integer program with n variables, $p \leq n$ of which are 0-1 variables

$$cx \rightarrow \max \quad x \in K^0 \quad (2.7)$$

where $K^0 := \{x \in \mathbb{R}_+^n : A^0x \leq b^0, x_i \in \{0, 1\}, i = 1, \dots, p\}$ is feasible set. Let the LP relaxation constraint set be

$$K := \{x \in \mathbb{R}^n : Ax \leq b\} \quad (2.8)$$

and assume that the system $Ax \leq b$ already contains the constraints $x_j \geq 0, i = 1, \dots, n$, and $x_j \leq 1, i = 1, \dots, p$. Since we are dealing with mixed integer program, we are interested in the convex hull of infinite set of points, we define $\text{conv}(K^0)$ as the closure of all finite convex combinations of points in K^0 . We describe the sequential convexification procedure:

The sequential convexification procedure

Step 1. Select an index $j \in \{1, \dots, p\}$.

Step 2. Multiply (every inequality of) $Ax \leq b$ with x_j and $1 - x_j$ to obtain the nonlinear system

$$\begin{aligned} x_j(Ax - b) &\leq 0 \\ (1 - x_j)(Ax - b) &\leq 0 \end{aligned} \quad (2.9)$$

Linearize the system (2.9) by substituting y_i for $x_i x_j, i = 1, \dots, n, i \neq j$, and replacing x_j for x_j^2 . Call the polyhedron defined from resulting system $M_j(K)$.

Step 3. Project $M_j(K)$ onto x -space by eliminating y_i variables. Call the resulting polyhedron $P_j(K)$.

The linearization in Step 3 yields, among others, the inequalities $y_i \geq 0$, $y_i \leq x_i$, for $i = 1, \dots, n$, and $y_i \leq x_j$, $y_i \geq x_i + x_j - 1$, for $i = 1, \dots, p$. Note that, if the system defining K has m constraints and n variables, then the system defining $M_j(K)$ has $2m$ constraints and $2n - 1$ variables.

The problem that remains in order to implement the procedure is to carry out the Step 3. Let

$$M_j(K) = \{(x, y) \in \mathbb{R}^q \times \mathbb{R}^r : Dx + By \leq d\}$$

where D and B are $m \times q$ and $m \times r$ matrices, respectively, and d is an m -vector. Then the projection of $M_j(K)$ onto the x -space can be described by

$$P_j(K) = \{x \in \mathbb{R}^q : (uD)x \leq (ud) \text{ for all } u \in C\},$$

where $C = \{u \in \mathbb{R}^m : uB = 0, u \geq 0\}$. Thus, the problem of finding a valid inequality in Step 3 of the procedure that cuts off a current fractional solution x^* can be solved by the linear program

$$\begin{aligned} u(Dx^* - d) &\rightarrow \max \\ u &\in C \end{aligned} \tag{2.10}$$

The following theorem shows that j -th component of each vertex of $P_j(K)$ is either 0 or 1.

Theorem 2.3.5 [16] $P_j(K) = \text{conv}(K \cap \{x \in \mathbb{R}^n : x_j \in \{0, 1\}\})$.

We can iterate the whole procedure. For $t \geq 2$, any sequence of indices $i_1, \dots, i_t \in \{1, \dots, p\}$ define $P_{i_1, \dots, i_{t-1}, i_t}(K) := P_{i_t}(P_{i_{t-1}} \dots (P_{i_1}(K)) \dots)$. In [16] it is shown that the convex hull of feasible solutions can be obtained by iterating the procedure p times.

Theorem 2.3.6 [16] For any $t \in \{1, \dots, p\}$,

$$P_{i_1, \dots, i_t}(K) = \text{conv}\left(K \cap \{x \in \mathbb{R}^n : x_{i_k} \in \{0, 1\}, k = 1, \dots, t\}\right).$$

Theorem 2.3.6 shows that the result does not depend on the order, in which the procedure is applied to the selected variables.

Corollary 2.3.7 $P_{1,\dots,p}(K) = \text{conv}(K^0)$.

Projecting $M_j(K)$ onto the x -space amounts to solve the LP (2.10), which is unbounded, since C is a polyhedral cone. For implementation issues, C is often truncated by some “normalizing set”. If in the lifting procedure the index j of binary variable x_j that attains fractional value in a feasible solution was chosen, then an optimal solution of the LP (2.10) cuts off x^* . The reader is referred to [16] for more explanations and details in constructing the lift-and-project algorithm for 0-1 mixed integer program.

Another way of performing the lift-and-project procedure is due to Lovász and Schrijver [104]. In this procedure the lifting onto the higher dimensional space is obtained by multiplying every inequality by every 0-1 variable and its complement, then linearizing the resulting system of quadratic inequalities and finally projecting back the system onto the original space.

Lovász-Schrijver procedure

Step 1. Multiply (every inequality of) $Ax \leq b$ with x_j and $1 - x_j$, $j = 1, \dots, p$, to obtain the nonlinear system

$$\begin{aligned} x_1(Ax - b) &\leq 0 \\ (1 - x_1)(Ax - b) &\leq 0 \\ &\vdots \\ x_p(Ax - b) &\leq 0 \\ (1 - x_p)(Ax - b) &\leq 0 \end{aligned} \tag{2.11}$$

Step 2. Linearize the system (2.11) by replacing y_{ij} for $x_i x_j$, setting $y_{ij} = y_{ji}$, $i = 1, \dots, n$, $j = 1, \dots, p$, $i \neq j$, and replacing x_j for x_j^2 , $j = 1, \dots, p$. Call the polyhedron defined from resulting system $M(K)$.

Step 3. Project $M(K)$ onto x -space. By eliminating y_{ij} variables as $y_{ij} = 0$, we obtain the project of $M(K)$. Call the resulting polyhedron $N(K)$.

The linearization yields, among others, the inequalities $y_{ij} \geq 0$, $y_{ij} \leq x_i$, for $i = 1, \dots, n$, $j = 1, \dots, p$ and $i \neq j$, and $y_{ij} \leq x_j$, $y_{ij} \geq x_i + x_j - 1$, for $i = 1, \dots, p$ and $j = 1, \dots, p$. Note that, if the system defining K has m constraints and n variables, of which p are 0-1

constrained in K^0 , then the system defining $N(K)$ has $2pm$ constraints and $pn+n-\frac{1}{2}p(p+1)$ variables.

We can iterate the whole procedure by denoting $N^1(k) := N(K)$ and $N^t(K) = N(N^{t-1}(K))$, for $t \geq 2$. Lovász and Schrijver [104] have shown that the convex hull of feasible solutions can be obtained by iterating the procedure p times.

Theorem 2.3.8 [104] $N(K) \subseteq \text{conv}(K \cap \{x \in \mathbb{R}^n : x_j \in \{0, 1\}\})$, for $j = 1, \dots, p$.

Theorem 2.3.9 [104] $N^p(K) = \text{conv}(K^0)$.

The third way, suggested by Sherali and Adams [147], is the following lift and project procedure.

Sherali-Adams procedure

Step 1. Multiply (every inequality of) $Ax \leq b$ with every product of the form $\left(\prod_{j \in J_1} x_j\right) \left(\prod_{j \in J_2} (1 - x_j)\right)$, where J_1 and J_2 are disjoint subsets of $\{1, \dots, p\}$ such that $|J_1 \cup J_2| = r$. Call the nonlinear system (NL_r) .

Step 2. Linearize (NL_r) by replacing x_j for x_j^2 , and replacing a variable w_J for every product $\prod_{j \in J} x_j$, where $J \subset \{1, \dots, p\}$, and v_{Jk} for every product $x_k \prod_{j \in J} x_j$ where $J \subset \{1, \dots, p\}$ and $k \in \{p+1, \dots, n\}$. Call the polyhedron defined by resulting system X_r .

Step 3. Project X_r onto the x -space. Call the resulting polyhedron K_r .

It is easy to see that $K^0 \subset K_p \subset \dots \subset K_1 \subset K$. Sherali and Adams showed that this procedure directly yields a linear description of $\text{conv}(K^0)$.

Theorem 2.3.10 [147] $K_p = \text{conv}(K^0)$.

The connection to the sequential convexification procedure is:

Theorem 2.3.11 [16] For $r = 1, \dots, p$, $K_r \subset P_{1, \dots, r}(K)$.

The lift-and-project procedure is closely related to the results of the disjunctive programming (DP) [14, 15], the optimization over unions of polyhedra. In fact, the Theorem 2.3.5 states that $P_j(K) = \text{conv}(P^0 \cup P^1)$ where $P^0 := K \cap \{x \in \mathbb{R}^n : x_j = 0\}$ and $P^1 := K \cap \{x \in \mathbb{R}^n : x_j = 1\}$. The inequalities obtained by projecting $M_j(K)$ onto x -space may be viewed as inequalities obtained from the disjunction of K into P^0 and P^1 . Therefore, lift-and-project is a specialization of DP. We will have a closer look into the issue.

Theorem 2.3.12 [15] Let $\Pi_i := \{x \in \mathbb{R}^n : A^i \leq b^i\}, \forall i \in Q$, be a finite set of nonempty polyhedra. Then $\text{conv}(\bigcup_{i \in Q} \Pi_i)$ is the set of points $x \in \mathbb{R}^n$ for which there exists vectors $(y^i, y_0^i), i \in Q$, such that

$$\begin{aligned} x - \sum_{i \in Q} y^i &= 0 \\ A^i y^i - b^i y_0^i &\leq 0 \quad \forall i \in Q \\ \sum_{i \in Q} y_0^i &= 1 \\ y_0^i &\geq 0 \quad \forall i \in Q. \end{aligned} \tag{2.12}$$

Here, we assume $\Pi_i \neq \emptyset, \forall i \in Q$. If $\Pi_k = \emptyset$ for some $k \in Q$, then the Theorem 2.3.12 is still valid if the following regularity condition holds:

$A^k y^k \geq 0$ implies that $y^k = \sum_{i \in Q^*} y^i$ for some $Q^* \subset Q \setminus \{k\}$ such that, $\forall i \in Q^*, \Pi_i \neq \emptyset$ and $A^i y^i \geq 0$.

The characterization of the convex hull of a union of polyhedra is contained in the following theorem, which will play an important role in the design of cutting plane algorithm in Section 2.4.1. The result is stated as it applied to $P_j(K) = \text{conv}(P^0 \cup P^1)$.

Theorem 2.3.13 [16, 15] $P_j(K) = \{x \in \mathbb{R}^n : \alpha x \leq \beta, \forall (\alpha, \beta) \in P_j^*(K)\}$, where $P_j^*(K)$ is the set of $(\alpha, \beta) \in \mathbb{R}^{n+1}$ for which there exist vectors $u, v \in \mathbb{R}^{m+n+p}$ and $u_0, v_0 \in \mathbb{R}$ satisfying

$$\begin{aligned} \alpha - uA - u_0 e_j &= 0 \\ \alpha &\quad -vA - v_0 e_j = 0 \\ ub &= \beta \\ v b + v_0 &= \beta v \\ u, v &\geq 0 \end{aligned} \tag{2.13}$$

where e_j is the j -th unit vector in \mathbb{R}^n .

Further, if K is a full dimensional polyhedron and $P^0 \neq \emptyset \neq P^1$, then for any constant $\beta_0 \neq 0$, $\alpha x \leq \beta_0$ defines a facet of $P_j(K)$ if and only if α is an extreme point of $P_j^*(K)_{\beta_0}$, the polyhedron obtained from $P_j^*(K)$ by setting $\beta = \beta_0$.

2.3.4. Lagrangian relaxation

Given an IP

$$cx \rightarrow \max, x \in X \quad (\text{IP})$$

where $X = \{x \in \mathbb{Z}_+^n : Ax \leq b\}$. Let $z(x) = \max\{cx, x \in X\}$ and z_{IP} be an optimal value. Suppose that we can partition the set of constraint inequalities into two subsets, a set of complicated inequalities $A^1x \leq b^1$ with $m_1 < m$ rows, and the inequalities $A^2x \leq b^2$ easy to solve: $A = \begin{pmatrix} A^1 \\ A^2 \end{pmatrix}$ and $b = \begin{pmatrix} b^1 \\ b^2 \end{pmatrix}$. Then dropping the complicated inequalities may lead to an easier problem $cx \rightarrow \max, x \in X^1$, where $X^1 = \{x \in \mathbb{Z}_+^n : A^2x \leq b^2\}$. To regard the dropped constraints, we add them with penalty parameters to the objective function.

$$z(\lambda, x) = cx + \lambda(b^1 - A^1x) \rightarrow \max, x \in Q \quad (\text{L})$$

where $Q = \{x \in \mathbb{Z}_+^n : A^2x \leq b^2\}$, $\lambda \in \mathbb{R}_+^{m_1}$.

A relaxation obtained through this way is called a *Lagrangian relaxation* of IP with respect to the constraints $A^1x \leq b^1$. Note that $b^1 - A^1x \geq 0$, $\forall x \in X$, therefore $z(\lambda, x) \geq z(x)$. The relaxation makes sense only if this problem is much easier to solve than the original problem. The use of Lagrangian methods in discrete optimization were proposed by Lorie and Savage [103], Everett [52] and Gilmore and Gomory [68]. However, the Lagrangian method received much attention through the implementing of a successful algorithm for the traveling salesman problem by Held and Karp [90], and were applied to many combinatorial optimization problems, including scheduling problems [59] and general IP [144, 60].

By choosing the value $\lambda \in \mathbb{R}_+^{m_1}$ the solution of (L) gives an upper bound to the optimal objective value of (IP).

$$z_L = \max_{x \in Q} z(\lambda, x), \quad z_L \geq z_{IP}$$

We would like to find the value of λ for the least upper bound. This leads to a dual problem:

$$z_D = \min_{\lambda \geq 0} z(\lambda, x) \quad (\text{D})$$

Most schemes for finding λ take as their objective to find the optimal or a near optimal solution to the above dual problem.

Proposition 2.3.14 $z_D = \max\{cx : A^1x \leq b^1, x \in \text{conv}(Q)\}$.

If for fixed λ , the Lagrangian relaxation becomes easy to solve, then the dual problem can

be used to get a good lower bound on the (IP) objective. Since assuming that the feasible set $Q = \{x \in \mathbb{Z}_+^n : A^2x \leq b^2\}$ is finite, it can be represented as $Q = \{x^i, i \in K, r^j, j \in Y\}$, where $x^i, i \in K$ are the vertices of $\text{conv}(Q)$, and $r^i, i \in Y$ are the extreme rays of $\text{conv}(Q)$. This allows us to represent (D) as the following linear program

$$\begin{aligned}
 z_D &= \min w \\
 w &\geq cx^i + \lambda(b^1 - A^1x^i) \quad \forall i \in K \\
 \lambda A^1r^i &\geq cr^i \quad \forall i \in Y \\
 \lambda &\geq 0.
 \end{aligned} \tag{2.14}$$

The LP dual of the above problem is the following

$$\begin{aligned}
 z_D &= \max \sum_{i \in K} \alpha_i cx^i + \sum_{i \in Y} \beta_i cr^i \\
 \sum_{i \in K} \alpha_i A^1x^i + \sum_{i \in Y} \beta_i A^1r^i &\leq b^1 \\
 \sum_{i \in K} \alpha_i &= 1 \\
 \alpha_i &\geq 0 \quad \forall i \in K \\
 \beta_i &\geq 0 \quad \forall i \in Y.
 \end{aligned} \tag{2.15}$$

Both problems (2.14) and (2.15) have important impacts on designing the algorithms to solve (D). The problem (2.14) makes it apparent that $z(\lambda)$ is the upper envelope of finite linear functions, and therefore piecewise linear and convex. Minimization of piecewise linear function over nonnegativity constraint is a widely studied subject and the hill climbing methods can be applied. The subgradient method is widely developed in the subject to solve it.

For example, we recall the non-capacitated facility location problem (UFL) that is formulated in Section 2.1. The IP formulation of the problem is:

$$\sum_{i=1}^n f_i x_i + \sum_{i=1}^n \sum_{j=1}^m c_{ij} y_{ij} \rightarrow \min \tag{UFL}$$

$$y_{ij} \leq x_i \quad \forall i \in \{1, \dots, n\}, \forall j \in \{1, \dots, m\} \quad (2.16)$$

$$\sum_{i=1}^n y_{ij} = 1 \quad \forall j \in \{1, \dots, m\} \quad (2.17)$$

$$\begin{aligned} x_i &\in \{0, 1\} & \forall i &\in \{1, \dots, n\} \\ y_{ij} &\in \{0, 1\} & \forall i &\in \{1, \dots, n\}, \forall j \in \{1, \dots, m\} \end{aligned} \quad (2.18)$$

The Lagrangian relaxation of UFL with respect to the constraints (2.17) is

$$\sum_{i=1}^n f_i x_i + \sum_{j=1}^m \sum_{i=1}^n (c_{ij} + \lambda_j) y_{ij} - \sum_{j=1}^m \lambda_j \longrightarrow \min \quad (\text{UFL-L})$$

$$\begin{aligned} y_{ij} &\leq x_i & \forall i &\in \{1, \dots, n\}, \forall j \in \{1, \dots, m\} \\ x_i &\in \{0, 1\} & \forall i &\in \{1, \dots, n\} \\ y_{ij} &\in \{0, 1\} & \forall i &\in \{1, \dots, n\}, \forall j \in \{1, \dots, m\}. \end{aligned}$$

2.4. Cutting plane algorithms

Since the LP is polynomially solvable, most approximation algorithms for IP are based on the idea of successively construct LP relaxations. The cutting plane algorithm deals with successively tighter LP relaxations, hopefully, to find integer solution. The idea is to relax the IP into the LP, solve this LP; and if solution to the LP, say x^0 , is not integral then add the inequality into constraint set, which is valid for IP's feasible set, but not valid for x^0 , therefore cuts off the non-integral solution x^0 .

The implementation of this basic idea leads into different methods how to construct these cut inequalities. Cutting plane methods were the first systematic technique for the solution of ILPs. Dantzig, Fulkerson and Johnson [40] first proposed the cutting plane approach by successfully solving an example of a large scale traveling salesman problem, and directed researchers' attention to the solving of ILPs. Gomory [81, 80] gave a cutting plane method that guaranteed integer solution in a finite number of steps. We mentioned this algorithm in Section 2.3.1. There are various problem specific valid inequalities and inequality generation methods. The efficient method is to construct or choose valid inequalities, which define, if possible the facets, or at least the faces of the convex hull of the constraint set of IP.

Since IP is NP-hard, to represent the convex hull of the feasible set X , we need in general exponentially many inequalities.

Let F be a family of valid inequalities for $\text{conv}(X)$ in the form $\alpha x \leq \beta$, $(\alpha, \beta) \in F$. The cutting plane procedure can be defined as following.

The basic procedure: Solve the LP relaxation. If the solution x^0 is non-integral then find the $(\alpha^0, \beta^0) \in F$ that is not valid for x^0 , $\alpha^0 x^0 > \beta^0$, and add it into constraints to get tighter LP. If solution of LP is integral then we are done; otherwise these steps are iterated.

In practice, to continue this procedure until to find an integral solution might need much time, therefore, one stops the procedure when the integral gap is small enough or the difference between the objective values of two iterations is small enough.

The problem of finding an element of F that cuts off a point x^0 is hard. Given a family of inequalities F and a fractional point x^0 , the problem of finding an inequality in F that is violated by x^0 or showing that no such inequality exists is called a *separation problem* for polyhedron defined by the family of inequalities F . For the integer programs this problem is not solvable in polynomial time in general. However, there are some special cases where the polyhedron separation problem is easily solvable. There are also fast approximation algorithms and heuristics that deal with the separation problem.

It is difficult to construct the cutting inequalities for general IP. For this reason we have to study the underlying constraint set X , and specifically develop the methods to generate the cutting inequalities for X . However, in many IP applications the cover inequalities for the knapsack problem are promising inequalities that lead to successful cutting planes. In the following we consider the valid inequalities for knapsack problem. We describe the valid inequalities for knapsack problem and the cutting plane algorithm based on these inequalities accordingly the description in [1].

Assume the constraint set of a 0 – 1 knapsack problem

$$X = \{x \in \{0, 1\} : \sum_{j \in N} a_j x_j \leq b\} \quad (2.19)$$

where $N = \{1, \dots, n\}$, $a_j \in \mathbb{Z}_+$, $\forall j \in N$ and $b \in \mathbb{Z}_+$. We assume $a_j \leq b$, $\forall j \in N$. For convenience, let us order $a_1 \geq a_2 \geq \dots \geq a_n$. Let $C \subset N$ represent the index set of “1” components of the vector x . Accordingly, let x^C denote the 0–1 vector that the components with indices C are 1’s, and other components are 0’s. A set C is called *cover* if $\sum_{j \in C} a_j > b$,

i.e., $x^C \notin X$.

Proposition 2.4.1 [13, 119, 160] If C is a cover then

$$\sum_{j \in C} x_j \leq |C| - 1 \quad (2.20)$$

is a valid inequality for X .

The inequality (2.20) is called *cover inequality*. The cover is minimal if all of its subsets are not cover. Note that, if a cover I is not minimal, then $\sum_{j \in I} x_j \leq |I| - 1$ is the sum of $\sum_{j \in I'} x_j \leq |I'| - 1$ and $x_j \leq 1$ for $j \in I \setminus I'$, where I' is minimal cover. The extension of cover C , $E(C)$, is the set $C \cup \{k \in N \setminus C : a_k \geq a_j, \forall j \in C\}$. Note that if $I \subset E(C)$ and $|I| \geq |C|$ then $x^I \notin X$. Therefore, we can say:

Proposition 2.4.2 [13, 119] If C is a minimal cover then

$$\sum_{j \in E(C)} x_j \leq |C| - 1 \quad (2.21)$$

is a valid inequality.

Proposition 2.4.3 Let $C = \{j_1, \dots, j_r\}$ be a minimal cover with $j_1 < \dots < j_r$. If any of the following conditions holds, then (2.21) gives a facet of $\text{conv}(X)$:

- $C = N$;
- $E(C) = N$ and (i) $C \setminus \{j_1, j_2\} \cup \{1\}$ is not a cover;
- $C = E(C)$ and (ii) $C \setminus \{j_1\} \cup \{k\}$ is a cover, where $k = \min\{j : j \in N \setminus E(C)\}$;
- $C \subset E(C) \subset N$ and (i) and (ii).

From this proposition, the following observation can be made.

Corollary 2.4.4 If C is a minimal cover for X and (C_1, C_2) is any partition of C with $C_1 \neq \emptyset$, then $\sum_{j \in C_1} x_j \geq |C_1| - 1$ gives a facet of $\text{conv}(X(C_1, C_2))$, where $X(C_1, C_2) = X \cap \{x \in \{0, 1\}^n : x_j = 0 \text{ for } j \in N \setminus C, x_j = 1 \text{ for } j \in C_2\}$.

Furthermore, we derive from the cover inequalities another valid inequalities, the *lifted cover inequalities* (LCIs) [17, 119, 160]. By lifting up all the variables $j \in N \setminus C$ (i.e., considering $x_j = 1$), and lifting down all the variables $j \in C_2$ (i.e., considering $x_j = 0$) we obtain a facet defining lifted cover inequality for the $\text{conv}(X)$.

Proposition 2.4.5 *If C is a minimal cover for X and (C_1, C_2) is a partition of C with $C_1 \neq \emptyset$, then $\text{conv}(X)$ has a facet represented by LCI*

$$\sum_{j \in C_1} x_j + \sum_{j \in N \setminus C} \alpha_j x_j + \sum_{j \in C_2} \gamma_j x_j \leq |C_1| - 1 + \sum_{j \in C_2} \gamma_j \quad (2.22)$$

where $\alpha_j \geq 0, \forall j \in N \setminus C$, and $\gamma_j \geq 0, \forall j \in C_2$.

Note, that lifting-up is used to strengthen the cover inequality, since $\alpha_j = 0$ suffices for validity, and lifting-down is used to ensure validity, since $\gamma_j = 0$ does not yield a valid inequality. The special case occurs when we take $C_2 = \emptyset$ and $C = C_1$, then the resulting inequalities are called *simple* LCI:

$$\sum_{j \in C} x_j + \sum_{j \in N \setminus C} \alpha_j x_j \leq |C| - 1. \quad (2.23)$$

In the separation problem for LCIs the $C \subseteq N$ is unknown and given a $x^* \in \mathbb{R}^n \setminus \{0, 1\}^n$ (non-integral), we want to find a C (assuming that one exists) with $\sum_{j \in C} a_j > b$ and $\sum_{j \in C} x_j^* > |C| - 1$. Let $z \in \{0, 1\}^n$ be the characteristic vector of the cover C . Then such a cover can be found, or shown not to exist, by solving the following problem:

$$\zeta = \min \left\{ \sum_{j \in N} (1 - x_j^*) z_j : \sum_{j \in N} a_j z_j \geq b + 1, z \in \{0, 1\}^n \right\}. \quad (2.24)$$

Proposition 2.4.6 *Let ζ and C be the optimal solution of separation problem for LCI (2.24). Then*

- 1) *If $\zeta \geq 1$, then x^* satisfies all the cover inequalities for X ;*
- 2) *If $\zeta < 1$, then $\sum_{j \in C} x_j \leq |C| - 1$ is the most violated cover inequality for X and it is violated by the value of $1 - \zeta$.*

The separation problem (2.24) is a knapsack problem, which is NP-hard. In practice (2.24) is solved by fast heuristics approximately. The further details on computation of LCIs and the experimental studies are given in [83, 84].

2.4.1. Cutting plane algorithms for 0-1 IPs

We present the cutting plane algorithm using lifted cover inequalities for general 0-1 integer programming problem. Consider

$$\begin{aligned} \sum_{j \in N} c_j x_j &\rightarrow \max \\ \sum_{j \in N} a_{ij} x_j &\leq b_i \quad i = 1, \dots, m \\ x &\in \{0, 1\}^n \end{aligned} \tag{BIP}$$

Without loss of generality, assume that $a_{ij} \geq 0$, for all $i = 1, \dots, m$, $j = 1, \dots, n$, and $b_i \geq 0$, $i = 1, \dots, m$, i.e., the elements of constraint matrices are not negative. (If $a_{ij} < 0$, by complementing variables we replace $x_i = 1 - y_i$ and $a'_{ij} = -a_{ij}$, $b'_i = b_i - \sum_{i: a_{ij} < 0} a_{ij}$) Therefore every row of the constraints can be viewed as knapsack inequalities. This motivates the use of the LCIs in the cutting plane algorithm.

The separation problem for LCI is solved in two phases. First, we try to find the most violated cover inequality, then in the second phase we lift the identified cover inequality regardless of whether it is violated. Even if the cover inequality is valid, the LCI can be violated. Recall that lifted cover inequalities are of the form

$$\sum_{j \in C_1} x_j + \sum_{j \in N \setminus C} \alpha_j x_j + \sum_{j \in C_2} \gamma_j x_j \leq |C_1| - 1 + \sum_{j \in C_2} \gamma_j \tag{2.25}$$

where C is a minimal cover, $C_1 \cup C_2 = C$ and $C_1 \cap C_2 = \emptyset$. The coefficients (α_j) and (γ_j) can be chosen such that (2.25) defines a facet of the knapsack convex hull. Padberg [118, 119] proposed a recursive procedure for calculating the coefficients – a *sequential lifting* procedure [160, 165].

The performance of the cutting plane algorithm based on LCIs depends on the choice of the lifting sequence, since the different lifting sequences lead to different inequalities. Gu, Nemhauser and Savelsbergh [84] have shown that given a minimal cover, the problem of identifying a lifting sequence that leads to the most violated LCI is \mathcal{NP} -hard even for the simple LCIs.

Let $x^* \in \mathbb{R}^n$ be a nonintegral optimal solution to the LP relaxation of (BIP) and $L = \{j \in N : x_j^* = 0\}$ and $U = \{j \in N : x_j^* = 1\}$. Since the lifting coefficients for

the variables x_j , $\forall j \in L \cup U$, have no effect on the violence of the LCI, the integral valued variables are lifted after the fractional valued variables. There are several methods available for ordering the fractional variables. A natural one is using the order of nonincreasing absolute difference between current LP value and projected value, because the larger this difference is, there is more effect on violation. Another option is to lift them in order of nondecreasing magnitude of reduced costs [93]. The logic behind this sequence is that variables with a reduced cost of small magnitude are more important, at least locally, than variables with a reduced cost far away from 0. Yet another option is an adaptive greedy order [132], which only applies to fractional variables that have to be lifted up. In each step, the variable with the highest contribution to the left hand side of the LCI is lifted, i.e., $\alpha_j x_j^*$ is computed for each $j \in N \setminus C$, that is not yet lifted up and the variable for which $\alpha_j x_j^*$ is maximum is selected.

The efficient computation of lifting coefficients has an important role in the use of LCIs. Given a lifting sequence of variables in $N \setminus C_1$, the lifting coefficients can be computed by solving a series of related 0-1 knapsack problems [17]. The computational aspects of determining the lifting coefficients for cover inequalities have been studied, especially for simple LCIs. The lifting coefficients can be determined approximately or exactly due to algorithm design and desired computational time. Some algorithms compute the lifting coefficients exactly [132, 113]. The underlying 0-1 knapsack problem can be solved by dynamic programming efficiently, because of the small size of the coefficients. Some algorithms compute the coefficients approximately [38].

The best known algorithm that computes the lifting coefficients exactly uses dynamic programming to solve a reformulation of the lifting knapsack problem in which the roles of the objective and constraints are reversed [113, 165]. By this dynamic programming algorithm, computing all the lifting coefficients takes $O(|C|n)$ time for simple LCIs, and $O(|C|n^3)$ time for LCIs, if the fractional variables are lifted first, the variables with 1 values are lifted next, then the variables with value 0 are lifted at last.

Now we shortly outline the cutting plane algorithm with LCIs as follows. As an initial LP relaxation constraint set we take $X_R^1 = \{x \in \mathbb{R}_+^n : Ax \leq b, x \leq 1\}$.

Set $t = 1$.

Iteration t :

Step 1. Solve the relaxation $z_R^t = \max\{cx : x \in X_R^t\}$, and let x^t be an optimal solution.

Step 2. Optimality test: Stop if stopping criterion satisfies.

- Step 3.* ■ 3.1. For each row of the constraints $\sum_{j \in N} a_{ij}x_j \leq b_i$, $i = 1, \dots, m$, solve the separation problem that restated as knapsack problem to obtain a cover C . (cover inequality may not be violated by x^t);
- 3.2. Lift the cover inequality:
- a. Using some predefined lifting sequence compute the lifting coefficients.
 - b. If the resulted inequality is violated by x^t then goto step (c).
 - b.1. Else, choose $k = \arg \max_{j \in C} a_{ij}x_j^t$. Set $C_2 = \{k\}$ and by lifting procedure generate a facet defining inequality for $\text{conv}(X(k))$ from the cover $C \setminus \{k\}$, where $X(k) = \{x \in \{0, 1\}^{n-1} : \sum_{j \in N \setminus \{k\}} a_{ij}x_j \leq b_i - a_{ik}\}$.
 - b.2. Convert this inequality into a facet defining inequality for $\text{conv}(X)$ of the form (2.25) by lifting back in the variable x_k .
 - b.3. Check the resulted inequality for violation. Steps (b.1)–(b.2) can be repeated for different choices of k until violation yields or stopping criterion satisfies.
 - c. Let resulting LCIs be $\pi^i x^t > \pi_0^i$.

Step 4. Add the obtained inequalities into constraint set

$$X_R^{t+1} = X_R^t \cap \{x \in \mathbb{R}_+^n : \pi^i x \leq \pi_0^i\}.$$

Set $t \leftarrow t + 1$.

Another cutting plane algorithm for 0-1 mixed integer programs is derived from the lift-and-project approach which is discussed in Section 2.3.3. Setting the index j in sequential convexification procedure as the index of component of fractional solution x^* , and using the “normalization set” in the corresponding linear program, the valid inequality that cuts off the x^* yields. Recall the 0-1 mixed integer program (2.7)

$$cx \rightarrow \max \quad x \in K^0 \tag{MIP}$$

where $K^0 = \{x \in \mathbb{R}_+^n : A^0 x \leq b^0, x_j \in \{0, 1\}, j = 1, \dots, p\}$, and let $K = \{x \in \mathbb{R}_+^n : Ax \leq b\}$ be LP relaxation constraint set. For the cuts the facets of $P_j(K)$ are used. We

generate inequalities $\alpha x \leq \beta$ such that (α, β) is an extreme ray of the cone $P_j^*(K)$ of the Theorem 2.3.13. This can be done by solving the LP

$$\max\{a\alpha + b\beta : (\alpha, \beta) \in P_j^*(K) \cap S\} \quad (2.26)$$

where $(a, b) \in \mathbb{R}^{n+1}$ is a vector that determines the direction of the cut, $P_j^*(K)$ is the polyhedral cone defined by (2.13), and here S is a “normalization” set, which is aimed to truncate the cone $P_j^*(K)$. The cutting plane algorithm [16] is outlined below.

Step 1. Set $t \leftarrow 1$. $K^1 \leftarrow K = \{x \in \mathbb{R}^n : Ax \leq b\}$.

Step 2. Find $x^t \leftarrow \arg \max_{x \in K^t} cx$. If $x_j^t \in \{0, 1\}$ for $j = 1, \dots, p$, then stop.

Step 3. For $j \in \{1, \dots, p\}$ such that $0 < x_j^t < 1$, find

$$a^t \alpha^j + b^t \beta^j := \max\{a^t \alpha + b^t \beta : (\alpha, \beta) \in P_j^*(K^t) \cap S\}.$$

Step 4. Define K^{t+1} by adding to the constraints of K^t the cuts $\alpha^j x \leq \beta^j$ generated in Step 3.

Step 5. Set $t \leftarrow t + 1$ and go to Step 2.

Cutting Plane Algorithm For Maximum Cut Problem

Let us consider the MAX-CUT problem on a given graph $G = (V, E)$ with a weight function on the edge set $c : E \rightarrow \mathbb{R}_0^+$. We consider an edge-model of the MAX-CUT. Let $y \in \mathbb{R}^{|E|}$ denote the incidence vector of the cut: $y_e = 1$ if an edge e is on the cut, and $y_e = 0$ otherwise. We write y^S when y represents the cut S . A *cut polytope* $P^{cut}(G)$ of the graph G is a convex hull of edge-characteristic vectors of all cuts:

$$P^{cut}(G) := \text{conv}\{y^S \mid S \subseteq E\} \subset \mathbb{R}^{|E|}.$$

As introduced in [20], we assume following inequalities

$$0 \leq y_e \leq 1 \text{ for all } e \in E \quad (2.27)$$

and

$$\sum_{e \in F} y_e - \sum_{e \in C \setminus F} y_e \leq |F| - 1 \text{ for each cycle } C, \quad F \subseteq C, |F| \text{ odd.} \quad (2.28)$$

Note that, for each cycle C , $\sum_{e \in C} y_e$ is an even number. Barahona and Mahjoub [20] showed when the above inequalities define the facets of $P^{cut}(G)$.

Theorem 2.4.7 ([20])

- (I) An inequality (2.28) defines a facet of $P^{cut}(G)$ if and only if C is a chordless cycle.
- (II) The inequalities (2.27) define facets of $P^{cut}(G)$ if and only if e does not belong to any triangle of G .

To design a cutting plane algorithm for the MAX-CUT problem by incorporating the inequalities (2.28), there must exist an efficient method to solve the separation problem. We describe a polynomial algorithm [20] to solve the separation problem for inequalities (2.28). Let us write these inequalities as

$$\sum_{e \in C \setminus F} y_e + \sum_{e \in F} (1 - y_e) \geq 1 \quad \text{for a cycle } C, \quad F \subseteq C, |F| \text{ odd.}$$

For a given non-integral y we are looking for a minimum weighted cycle such that some edges have the weight y_{ij} , and an odd number of edges have the weight $1 - y_{ij}$. From given graph G we construct a new graph G' by assigning two nodes i' and i'' for every node i of G . For every edge $e = (i, j)$ of G we draw edges $(i'j')$ and $(i''j'')$ with weight y_{ij} , and edges $(i'j'')$ and $(i''j')$ with weight $1 - y_{ij}$. For node i we find the shortest path between i' to i'' on G' . By taking the minimum over all nodes of the lengths of corresponding shortest path, we find the weight of the cycle that we are looking for. Now, if the inequality (2.28) for the resulted cycle is not valid at y , then this inequality cuts off y . The separation problem is efficiently solvable since the shortest path is polynomially solvable.

Through cutting the relaxed polyhedron only by inequalities of type (2.28) we might end up with nonintegral solution \hat{y} because the complete description of inequalities is not known to describe the $P^{cut}(G)$ and there is no efficient algorithm to generate further facet defining or valid inequalities. Barahona and Mahjoub [20] showed that the cut polytope of the graph not contractible to K_5 is completely describable by the inequalities (2.27) and (2.28), by incorporating Seymours "Sums of Circuits Property" [143].

Theorem 2.4.8 ([20]) *A graph G is not contractible to K_5 if and only if $P^{cut}(G)$ is defined*

by following inequalities:

$$0 \leq y_e \leq 1, \text{ for each edge } e \text{ that does not belong to a triangle,}$$

$$\sum_{e \in F} y_e - \sum_{e \in C \setminus F} y_e \leq |F| - 1 \text{ for each chordless cycle } C, F \subseteq C, |F| \text{ odd.}$$

An important corollary of this theorem is:

Corollary 2.4.9 *The MAX-CUT problem is solvable in polynomial time on graphs not contractible to K_5 .*

2.5. Branch and Bound Algorithms

One of the mostly used and very strong methods to solve integer programs is branch and bound approach. Branch and bound (B&B) algorithms are widely used to solve the optimization problems, including not only the integer programming problems, but also nonlinear optimization problems. An early development of branch and bound algorithms is given by Land and Doig [101] on applications to integer programming and mixed integer programming. Dakin [39] modified Land and Doig's algorithm to make it easier for computer implementation and made it possible to apply for nonlinear mixed integer programs.

Branch and bound method divides the given problem into subproblems that are efficiently solvable. In other words, it partitions the solution space into the collection of subspaces and search for solution in the subspaces with the hope that it performs the search only in fraction of the collection until it finds the optimal or suboptimal solution. For integer programming application we can describe two procedures of branch and bound method as follows.

Branching process partitions the continuous solution space into subspaces, with an intention to exclude the part of continuous space that violates the (some of) integer constraints. This exclusion is done by introducing new constraints that are necessary to produce integer constraints, but by not losing the feasible integer solutions. In other words, the resulting collection of subproblems' feasible solutions cover all of the feasible solutions of primary problem.

Bounding process produces the decision: whether the subproblems should be split further (the solution subspaces should be concerned further). Each of the subproblems' optimal solutions gives an upper bound for optimal objective value. If this upper bound is lower than an objective value of the existing feasible solution, then this subproblem and the corresponding subspace are no more interesting for the search.

We describe the Dakin's algorithm [39] for (mixed) integer programs. Given an IP

$$cx \rightarrow \max, x \in X \quad (2.29)$$

where $X = \{x \in \mathbb{Z}_+^n : Ax \leq b\}$. Let $z(x) = \max\{cx, x \in X\}$ and z_{IP} be an optimal value. The algorithm starts by solving the LP relaxation

$$cx \rightarrow \max, x \in P \quad (2.30)$$

where $P = \{x \in \mathbb{R}_+^n : Ax \leq b\}$. Let x^0 be an obtained optimal solution of (2.30). In a case that x^0 is integral, so this is also optimal for (2.29). In a case that x^0 is fractional, then we consider a partitioning of the solution space. There exists at least one component (variable) of x^0 , which is fractional. Let x_j^0 be a fractional component. Then to escape from the LP-solution that yields non-integral value for the variable x_j , we consider two constraints

$$x_j \leq \lfloor x_j^0 \rfloor \text{ and } x_j \geq \lfloor x_j^0 \rfloor + 1.$$

By adding each of these constraints into the constraint set of (2.30), we obtain two subproblems

$$cx \rightarrow \max, x \in P \cap \{x_j \leq \lfloor x_j^0 \rfloor\} \quad (2.31)$$

and

$$cx \rightarrow \max, x \in P \cap \{x_j \geq \lfloor x_j^0 \rfloor + 1\}. \quad (2.32)$$

We solve these subproblems (LPs) and repeat the procedure for each of the two solutions so obtained. The iteration of algorithm continues with depth-first search. This is because the subproblem differs from its parent problem by only one constraint on one variable. Therefore, previous LP-solution can be used for dual simplex restart to solve the subproblems quickly.

The bounding process is simple. If LP relaxation delivers integral optimal solution x^k

then we do not divide this subproblem further. If an objective value at x^k is greater than an objective value at the current incumbent \hat{x} , $cx^k > c\hat{x}$, then $\hat{x} = x^k$. If at any iteration of the algorithm LP relaxation delivers optimal solution, of which the objective value is less than the objective value at current incumbent, then this LP relaxation is no more partitioned. The algorithm stops when no more subproblems to be divided, and the current incumbent is returned as optimal solution to the primary problem (2.29).

B&B for Maximum Satisfiability Problem

In the following we consider a branch and bound algorithm for the weighted maximum satisfiability problem. Since the variables can take only false or true (0 or 1) values we divide the search space into subspaces by setting the value of an underlying (branching) variable into 0 or 1. Assume that we are given a CNF-formula F with m clauses on a set of n variables X and a positive weight $w_C \in \mathbb{R}^+$ associated to each clause C of F . Let $X = \{x_1, \dots, x_n\}$. We are looking for a truth assignment T^* that maximizes the total weight of satisfied clauses

$$W(T^*) = \sum_{\substack{C \in F: \\ T \text{ satisfies } C}} w_C \longrightarrow \max$$

We can formulate the objective possibly by minimizing the sum of weights of the unsatisfied clauses.

The set of feasible solutions or the search space is the set of vertices of an n -dimensional hypercube $\{x : 0 \leq x \leq 1\} \subset \mathbb{R}^n$. At the start of the algorithm none of the variables' values are fixed. Then at each step of the algorithm we find a partial assignment by assigning a value to some variable. By fixing a value of some variable – assigning a value 0 or 1 – we create two subproblems, in other words, we search in the unique cubes of one dimension lower than the previous cube. If we assign the value 0 or 1 to each of variables, then we find a feasible solution. So, the branching process is simple.

The bounding process is an essential factor for the branch and bound algorithm. For the MAX-SAT problem we can give also a simple bounding process. There are fast heuristics available for the MAX-SAT problem [141, 140, 122, 131, 87]. With such heuristic methods we can obtain a lower bound to the optimal objective value and an upper bound to the weight of unsatisfied clauses in the optimal solution. Even, any random assignment delivers

these bounds to the optimal solution value. Identifying a “good” solution before starting the search will help us to avoid the examining of the non-promising sub-spaces of the search space. For some solution x let $u(x)$ denote the weight of unsatisfied clauses at x . Let \hat{x} denote the current incumbent solution. If the weight of the unsatisfied clauses that consist of only the literals those were already assigned the values is more than $u(\hat{x})$ – the weight of unsatisfied clauses of the best assignment so far – then the extending this partial solution will not yield a better solution. Therefore, we do not divide this subset of the search space further. The initial solution found by some fast heuristic or by a random assignment delivers the first trivial bound to the weight of unsatisfied clauses. When the 0 or 1 values assigned to all variables, we have new incumbent solution, and we update the $u(\hat{x})$.

Branch and bound algorithms for the maximum satisfiability problem are presented in many literatures, including Hansen and Jaumard [86], Wallace and Freuder [159], Niedermeier and Rossmanith [115], Borchers and Furman [30], Hirsch [92], Alsinet, Manja and Planes [3] and Shen and Zhang [145, 146]. The most of these algorithms incorporate the variants of the Davis-Putnam-Logemann-Loveland (DPLL) backtracking procedure [41], that implicitly enumerates all possible truth assignments for the satisfiability problem. These algorithms differ from each other first by branching process, in which the variable is chosen for the branching, and second by the bounding process, in which the bounding functions are incorporated. These factors are essential on yielding efficient and fast algorithm. Some algorithms use preprocessing procedure to intend the search set become substantially small. Here we want to mention that these branch and bound algorithms need much time to finish in order to find an optimal solution. In the experimental studies of the state of the art branch and bound algorithms [145] it is reported that these algorithms required long computing time to the MAX-2-SAT instances with up to 200 variables and 3500 clauses. Even to some instances there was no optimal solution found after exhaustive computing time (2 hours). The reported experiments were carried out in the similar computing environment to one, in which our experimental study of the linear programming based heuristic method were performed (Section 5.3).

2.6. Branch and Cut Algorithms

The term “branch and cut” was coined by Padberg and Rinaldi [117] that they proposed the algorithm for the TSP. A branch and cut (B&C) algorithm [82] is an extension of branch

and bound approach that incorporates the cutting planes to tighten the search space. In branch and cut algorithm, in each generated sub-problem, additionally, a family of cutting planes are added into constraint set to even more tighten the LP-relaxation.

We start similarly to branch and bound approach by solving the LP relaxation of the integer program. If the LP optimal basis solution is nonintegral we add cutting planes to the LP constraint set that cut off this nonintegral solution. To result an efficient algorithm, the inequalities that define the facets of feasible set (polyhedron) – the convex hull of feasible solutions – are generated and added to the LP relaxations.

The cutting planes can be valid for only considering subset of feasible set or can be globally valid for all feasible solutions. But, the inequalities used or generated in branch and cut algorithms in the most cases are valid for the whole feasible set and simulates the polyhedral structure of IP. With this feature the branch and cut algorithm differs significantly from traditional cutting planes such as Gomory cuts [81].

Since the cutting planes or the facet defining inequalities are more polyhedral in nature, it is convenient to particularly consider the problem for which the branch and cut algorithm to be developed. We take a look in to an algorithm for the MAX-CUT problem proposed by Barahona and Ladányi [19]. Let us consider the edge-model of the MAX-CUT problem on a given weighted graph $G = (V, E)$ with a weight function $c : E \rightarrow \mathbb{R}_+$, for each edge $e \in E$ there is a nonnegative weight c_e is given. Let $y^S = \{y_e, \forall e \in E\}$ denote the characteristic vector of cut S : $y_e = 1$ if e on the cut – the nodes connected through e on the different sides of cut, $y_e = 0$ otherwise.

Our aim is to formulate the LP relaxations in the sub-problems that are tighter (closer) to the cut polytope $P^{cut}(G)$, the convex hull of all characteristic vectors of cuts, $P^{cut}(G) = \text{conv}(\{y^S | S \subseteq V\}) \subset \mathbb{R}^{|E|}$.

Recall the cycle inequality (2.28) that defines the facet of cut: polyhedron $P^{cut}(G)$.

$$\sum_{y \in F} y_e - \sum_{y \in C \setminus F} y_e \leq |F| - 1 \quad \text{for cycle } C \text{ and } F \subseteq C, |F| \text{ odd.} \quad (2.33)$$

Then the linear programming relaxation of the MAX-CUT problem is

$$\begin{aligned} \sum_{e \in E} c_e y_e &\longrightarrow \max \\ \sum_{y \in F} y_e - \sum_{y \in C \setminus F} y_e &\leq |F| - 1 \\ y_e &\geq 0 \\ y_e &\leq 1 \end{aligned}$$

where C is a cycle and $F \subseteq C$ has odd number of elements.

At the start we formulate an LP relaxation by introducing the inequalities (2.33) for some triangles of G . We solve the resulting LP. If the optimal basis solution of this LP yields to be an integral solution, then it is an optimal cut. Otherwise we generate more facet defining inequalities of type (2.33), which cut off the non-integral solution. In [19] a fast heuristic is applied to separate the nonintegral solution, instead of the polynomial time separation algorithm of [20]. The heuristic separation is as follows. Let \hat{x} be the non-integral point to be separated. We define the graph $G' = (V', E')$ identical to the G , but different on edge-weights. For each edge $e = (ij) \in E$, we draw an edge $e' = (i'j') \in E'$ with weight $c'_e = c_e \max(\hat{x}_e, 1 - \hat{x}_e)$. Then find a maximum weighted spanning tree T' of G' . We partition the edges of T' into two subsets T'_1 and T'_2 . For an edge $e' \in T'$, if the corresponding variable $\hat{x}_{e'} \geq 1 - \hat{x}_{e'}$ then the adjacent nodes of e' should be on different sides of the cut. We add these edges into the subset T'_1 . Otherwise, if $\hat{x}_{e'} < 1 - \hat{x}_{e'}$ then the adjacent nodes of edge e' should be on the same side of the cut. We add these edges into the subset T'_2 . By this partition the edges of T'_1 define a heuristic cut H . For an edge $e' \notin T'$ we add it into T' and consider the created cycle C . If $e' \in H$ then by setting $F = T'_1$ we test the violation of the inequality (2.33). If the inequality is violated, then we found the sets C and F . If $e' \notin H$ then we set $F = T'_1 \cup \{e'\}$. If we cannot generate facet defining inequality then we start the branching and divide the problem into sub-problems.

There are many considerations to be taken care to result the efficient implementations of branch and cut algorithms, such as which family of inequalities should be used, how many inequalities can be added into the constraint set and on which variable should be branched. Since there must be the LPs solved repeatedly, one consideration is to use fast heuristics to solve the LPs instead of using the simplex based method. For example, Barahona and Ladányi [19] proposed branch and cut algorithms for the problems of spanning tree on

graphs and maximum cut, which apply the volume algorithm [18] to the LP relaxations.

The surveys of the branch and cut algorithms can be found in Jünger et al. [96] and Caprara and Fischett [32].

For comprehensive understanding the subject of integer programming and the related theories in more detail, the reader is referred to the books by Nemhauser and Wolsey [113] and by Wolsey [161].

Chapter 3

State of the Art

In this chapter we review the computational aspects, the algorithmic results for the maximum cut and maximum satisfiability problems, and heuristic methods for solving them. Additionally, we briefly review the algorithmic results for the longest path problem. The review of the LP-based hybrid approaches, which are the hybridizations of LP and the metaheuristics, is the content of the last part of this chapter.

The reader is referred to Poljak and Tuza [128] for a comprehensive survey of the MAX-CUT problem. The comprehensive overviews of the MAX-SAT problem have been given by Hansen and Jaumard [86], and also by Battiti and Protasi [23].

3.1. Algorithmic Results for the Max-Cut and Max-SAT

Let P be an optimization problem (maximization) and each valid instance I of P comes with a nonempty set of feasible solutions and optimal objective value $OPT(I) > 0$. An algorithm \mathcal{A} , that finds for each instance I of P the solution with objective value at least $\rho OPT(I)$ (for some $\rho < 1$) in polynomial time, is called *ρ -approximation algorithm* for P and ρ is called *approximation ratio* of \mathcal{A} .

For each instance I of P and each $\epsilon > 0$, if an algorithm \mathcal{A} finds in polynomial time the solution with objective value at least $(1 - \epsilon) OPT(I)$, then \mathcal{A} is called *Polynomial Time Approximation Scheme* (PTAS).

A simple greedy algorithm gives $\frac{1}{2}$ -approximation for the MAX-CUT [136], thus $\frac{1}{4}$ -approximation for the DIRECTED MAX-CUT, and $\frac{1}{2}$ -approximation for the MAX-2-SAT [65].

For the SIMPLE MAX-CUT problem in connected graphs Poljak and Turzík [126] have designed an approximation algorithm, that finds a cut with at least $\frac{1}{2}m + \frac{1}{4}(n - 1)$ edges, with running time of $O(n^3)$. Ngoc and Tuza [114] have improved the running time bound as follows:

Theorem 3.1.1 [114] *For any connected graph, a cut with at least $\frac{1}{2}m + \frac{1}{4}(n - 1)$ edges can be found in $O(m)$ steps.*

For weighted case there is a following result.

Theorem 3.1.2 [126] *For every instance of the MAX-CUT problem with graph G and weight function c , a cut with weight at least*

$$\frac{1}{2} \sum_{e \in E} c_e + \frac{1}{4} \min_{T \subseteq E} \sum_{e \in T} c_e$$

can be found by an algorithm of running time $O(mn)$, where the minimum is taken over all spanning trees T of G .

Yannakakis [163] gave an $\frac{3}{4}$ -approximation algorithm for the MAX-2-SAT problem:

Theorem 3.1.3 [163] *The MAX-2-SAT can be approximated with ratio $\frac{3}{4}$ within running time of $O(m)$.*

Two decades after the proposition of the simple algorithm for the MAX-CUT problem it was still an open question how to improve the approximation constant factor $\frac{1}{2}$. A significant improvement on approximation ratios was achieved by Goemans and Williamson[79] using semidefinite programming (SDP), who gave algorithms with approximation ratio 0.79607 for the DIRECTED MAX-CUT and with approximation ratio 0.87856 for the MAX-2-SAT, respectively.

Theorem 3.1.4 [79] *There is a 0.87856-approximation algorithm for the nonnegative-weighted MAX-CUT problem.*

Delorme and Poljak [47, 46, 48] developed a theory of eigenvalue bounds and showed that eigenvalue minimization can be used to obtain upper bounds on the maximum cut. They showed that eigenvalue bound provides very good bound on the maximum cut (for

nonnegative weighted graphs) in practice and studied the worst case ratio between the eigenvalue bound and the maximum cut. The worst-case graph was a 5-cycle where the ratio is $\frac{32}{25+5\sqrt{5}} = 0.8844\dots$, but they were unable to prove the worst case ratio.

A lot of work has been done to improve the approximation ratios of Goemans and Williamson. Feige and Goemans [53] have improved the ratio to 0.859387 for the DIRECTED MAX-CUT problem and 0.931090 for the MAX-2-SAT problem using a rotation technique on the vectors (solutions) found by semidefinite programming relaxation. Zwick [166] achieved the ratios of 0.859643 and 0.931091 for the DIRECTED MAX-CUT and MAX-2-SAT problems, respectively. Matuura and Matsui [106, 107] have obtained ratios of 0.863 for the DIRECTED MAX-CUT and 0.935 for the MAX-2-SAT, using hyperplane separation technique with skewed distribution functions on the sphere. Lewin, Livnat and Zwick [102] have obtained algorithms with even better approximation ratios through combining and improving the techniques and results used and developed by Matuura and Matsui, and Feige and Goemans, of at least 0.874 for the DIRECTED MAX-CUT and at least 0.940 for the MAX-2-SAT problem.

For some graph classes these problems are efficiently solvable.

Theorem 3.1.5 [6] *There are PTASs for dense instances of the DIRECTED-MAX-CUT, $|A| = \Theta(|V|^2)$, and of MAX-2-SAT, $|F| = \Theta(|X|^2)$.*

On planar graphs, the MAX-CUT problem is polynomially solvable [85, 116]. Moreover, the MAX-CUT problem is polynomially solvable on graphs not contractible to K_5 [20] (see Corollary 2.4.9). The variation of the MAX-3-SAT problem where every clause consists of exactly 3 literals and the number of occurrences of each variable is exactly 3, is polynomially solvable [120].

3.1.1. Non-approximability Results

It is interesting to see how far could we go in order to increase the approximation ratios, in other words how close could we reach to 1.0. Nevertheless, the DIRECTED MAX-CUT and the MAX- k -SAT problems are \mathcal{APX} -complete [121]. Arora et al.[7, 8] showed that there is no PTAS for DIRECTED MAX-CUT unless $\mathcal{P} = \mathcal{NP}$.

Theorem 3.1.6 [7] *There exists no PTAS for DIRECTED MAX-CUT problem unless $\mathcal{P} = \mathcal{NP}$*

The following non-approximability results showed by Håstad [89].

Theorem 3.1.7 [89] *For any $\epsilon > 0$ and $k \geq 3$, it is \mathcal{NP} -hard to approximate*

- MAX-E2-SAT problem within a factor $\frac{21}{22} + \epsilon$;
- MAX-E k -SAT problem within a factor $1 - 2^{-k} + \epsilon$;
- SIMPLE MAX-CUT problem within a factor $\frac{16}{17} + \epsilon$;
- DIRECTED SIMPLE MAX-CUT problem within a factor $\frac{11}{12} + \epsilon$.

Crescenzi et al. [37] showed that a version of MAX-CUT problem, where each edge has a nonnegative integer weight, is as hard to approximate as the SIMPLE MAX-CUT. They also showed that the weighted MAX-E2-SAT problem, where clause-weights are nonnegative integers, is as hard to approximate as MAX-E2-SAT. Consequently, the non-approximability results in Theorem 3.1.7 hold for the nonnegative (integer) weighted versions of the specified problems.

The two natural ways to model the MAX-CUT problem are, (I) modeling the cut by using the variables defined on the node set V of the graph – a *node model*, and (II) defining the variables on the edge set E of the graph – an *edge model* [125]. We consider the edge model.

Let $y^S : E \rightarrow \{0, 1\}$ be an edge-characteristic vector of the cut $S \subset V$, i.e. $y_e = 1$ if and only if $e = (v, w)$ crosses the cut, otherwise $y_e = 0$. A *cut polytope* $P^{cut}(G)$ of the graph G is the convex hull of the edge-characteristic vectors of all cuts:

$$P^{cut}(G) := \text{conv}\{y^S \mid S \subset V\} \subset \mathbb{R}^E.$$

It is convenient to assume the extended weight function into complete graph, by setting the weights of edges that are not in (given) edge set to 0: $c_{vw} = 0$ for $(v, w) \notin E$. We define by P_n^{cut} the cut polytope $P^{cut}(K_n)$ of complete graph K_n . In fact, $P^{cut}(G)$ is a projection of P_n^{cut} . Then $y \in \mathbb{R}^{\binom{n}{2}}$ and the MAX-CUT problem can be formulated as

$$c^t y \longrightarrow \max, \quad y \in P_n^{cut} \subset \mathbb{R}^{\binom{n}{2}}.$$

Intractability of the problem lies on the fact that the complete description of the cut polyhedron $P^{cut}(G)$ through the linear inequalities is actually not known. However, the facet defining and valid inequalities have been studied [20]. For example, the triangle

inequalities are valid for cut polytope, and define all the facets of $P^{cut}(G)$ of which $|V| = 3$. Moreover, the triangle inequalities define the facets of P_n^{cut} .

The simple LP-relaxation of the problem is to optimize the MAX-CUT over *fractional polytope* $P^{met}(G)$, which is defined by the triangle inequalities:

$$\begin{aligned} y_{uv} &\leq y_{vw} + y_{uw} && \text{for all } u, v, w \in V \\ y_{uv} + y_{vw} + y_{uw} &\leq 2 && \text{for all } u, v, w \in V \\ y_{vw} &\leq 1 && \text{for all } v, w \in V \\ y_{vw} &\geq 0 && \text{for all } v, w \in V. \end{aligned}$$

Then we obtain a relaxation of the MAX-CUT problem

$$\sum_{1 \leq v < w \leq n} c_{vw} y_{vw} \longrightarrow \max_{y \in P^{met}(G)}. \quad (3.1)$$

Poljak and Tuza [127] have shown that the *integrality gap* of the relaxation (3.1) is $\frac{1}{2} + \epsilon$ for any $\epsilon > 0$, which is the minimum ratio of the maximum cut and the objective value at the optimal fractional solution, where the minimum is taken over all graphs G .

Since the MAX-CUT problem is \mathcal{APX} -complete, the question is how far can we increase the integrality gap, i.e., can we write an LP relaxation with higher integrality gap? De la Vega and Kenyon-Mathieu [45] have shown that the integrality gap for the MAX-CUT problem can not be increased more than $\frac{1}{2} + \epsilon$ after doing small numbers of Sherali-Adams lift-and-project rounds [148], analogously to the integrality gap $2 - \epsilon$ shown for the vertex cover problem [5].

Theorem 3.1.8 [45] *Let $\epsilon > 0$ and t be fixed. The integrality gap of LP relaxation obtained from $P^{met}(G)$ by doing the t rounds of Sherali-Adams lift-and-project is at most $\frac{1}{2} + \epsilon$.*

Note that, the set of inequalities derivable in $O(1)$ rounds can be exponentially large, but as long as it comes with a polynomial separation oracle, therefore tractable by the Ellipsoid method. For dense graphs, the integrality gap is $1 - \epsilon$, $\forall \epsilon > 0$.

3.2. Algorithmic Results for the Longest Path Problem

The longest path problem is well-known to be \mathcal{NP} -hard [98].

Theorem 3.2.1 [139] *Given a directed graph $G = (V, A)$ and vertices $s, t \in V$, finding the longest $s - t$ path is \mathcal{NP} -hard.*

Monien [111] gave an algorithm that finds path of length k in time $O(k!nm)$ if such a path exists in any undirected graph. His algorithm finds in Hamiltonian graphs paths of length $\Omega(\log n / \log \log n)$ in polynomial time. Karger, Motwani and Ramkumar [97] showed that in 1-tough graph (1-tough means that by removing any k vertices from the graph, the graph can not be decomposed into more than k connected components [97]) a path of length $\Omega(\log n)$ can be found in polynomial time and for Hamiltonian graphs a path of length k , $k \leq n$, can be found in $O(m + k^2!)$ time. Actually in [97] it was shown that there are 1-tough graphs which do not have paths longer than $\Omega(\log n)$, therefore the result is tight in that respect. Fürer and Raghavachari [62] gave the same result for Hamiltonian graphs, i.e., a path of length $\Omega(\log n)$ can be found in polynomial time.

Alon, Yuster and Zwick [2] presented tighter bounds by introducing a randomized method *color-coding*. Their algorithm finds the longest path if the longest path has size of $\Theta(\log n)$, else it finds a path of length $\Omega(\log n)$. Vishwanathan [156] presented an algorithm which finds a path of length $O((\log n / \log \log n)^2)$ for Hamiltonian graphs. Björklund and Husfeldt [28] presented an algorithm that finds a path of length $\Omega((\log k / \log \log k)^2)$ in a graph with longest path of length k , which gives the performance ratio of $O(n(\log \log n)^2 / \log^2 n)$. Gabow [63] presented an algorithm, that finds a cycle through a given vertex v of length $\exp(\Omega(\sqrt{\log \ell / \log \log \ell}))$ in an undirected graph in polynomial time. Here, ℓ is the length of the longest cycle through the given node v of degree 2. This suggests the same bound for the longest cycle, longest path between two given nodes, and longest path problems. Gabow's algorithm is based on the results of Björklund and Husfeldt [63].

The non-approximability results are mainly due to Karger, Motwani and Ramkumar [97]. They showed that there exists no constant factor approximation algorithm for the LONGEST PATH problem unless $\mathcal{P} = \mathcal{NP}$, even for the graphs with maximum degree of 4. Moreover, they showed that the longest path has no PTAS for Hamiltonian graphs. Bazgan, Santha and Tuza [25] showed that a similar result holds for an even smaller class of graphs: the LONGEST PATH is not constant factor approximable and has no PTAS in cubic Hamiltonian graphs unless $\mathcal{P} = \mathcal{NP}$.

De la Vega and Karpinski [44] showed that the LONGEST PATH is $\mathcal{MAX} - \mathcal{SNP}$ -hard for “dense” instances. More specifically, they showed that for the graphs with minimum degree $d|V|$, $0 < d < 1/2$, the LONGEST PATH is $\mathcal{MAX} - \mathcal{SNP}$ -hard and that the LONGEST

PATH has no PTAS when restricted to Hamiltonian graphs with minimum degree of $d|V|$, where $0 < d < 1/2$.

For the directed case of the problem, Björklund, Husfeldt and Khanna [29] showed that the LONGEST PATH problem can not be approximated within a factor of $n^{1-\epsilon}$ for any $\epsilon > 0$ unless $\mathcal{P} = \mathcal{NP}$. In particular, the result holds for directed graphs with bounded outdegree that contain a Hamiltonian cycle.

3.3. Theoretical Bound for the Max-2-SAT

For a certain class of the MAX-2-SAT instances the theoretical bounds to the expected optimal value of the objective are given. We describe this class of instances – the *random instances*.

Let $(n \in \mathbb{Z}^+, m \in \mathbb{Z}^+, X = \{x_1, \dots, x_n\}, Z = \{z_1, \dots, z_m\}, w_j = 1, j = 1, \dots, m)$ be an instance of the MAX-2-SAT problem (with uniform weights). Let \mathcal{Z} be the set of all clauses of length 2, where each clause is consisting of two distinct variables. The variables are $\{x_1, \dots, x_n, \bar{x}_1, \dots, \bar{x}_n\}$. As *random instance* we assume that the clauses z_1, \dots, z_m are distributed uniformly and independently in \mathcal{Z} . We consider the case $\frac{m}{n} \rightarrow c$, c constant, as $n \rightarrow \infty$. Let random variable $f^*(n, m)$ denote the optimal solution of the random instance of size (n, m) .

The following asymptotic bounds for the expectation of value of random variable $f^*(n, m)$ are given by Coppersmith et. al. [35]. Before giving the statement, note that the notation $f(n) \lesssim g(n)$ indicates that f is less than or equal to g asymptotically: $\lim_{n \rightarrow \infty} \sup \frac{f(n)}{g(n)} \rightarrow 1$.

Theorem 3.3.1 [35] *For c large,*

$$\left(\frac{3}{4}c + (1 - o_c(1))\sqrt{c}\frac{\sqrt{8}-1}{3\sqrt{\pi}}\right)n \lesssim \mathbb{E}(f^*(n, cn)) \lesssim \left(\frac{3}{4}c + \sqrt{c}\sqrt{\frac{3\ln(2)}{8}}\right)n. \quad (3.2)$$

In Theorem 3.3.1, the factor $1 - o_c(1)$ indicates the value which is arbitrary close to 1 for all sufficiently large c . The constants $\frac{\sqrt{8}-1}{3\sqrt{\pi}}$ and $\sqrt{\frac{3\ln(2)}{8}}$ are approximately 0.343859 and 0.509855, respectively.

3.4. Heuristic Methods

Heuristic methods are a common tool to solve hard combinatorial optimization problems within reasonable time and with reasonable approximation results. Many efficient heuristic methods are based on local search. When the local search reaches local optima or plateaus, these methods use techniques that help the search to escape from the local optima or from the plateaus. Some of the techniques lead to the continuation of the search from a random point, or to the continuation of the search from some constructed points, or avoid to visit some of the points with attributes that are found and learned in the search history. Even though these strategies are called and defined differently in heuristics, the search strategies can be categorized into several classes: multi-start, memory based, variable neighborhood, population based and randomized. Some heuristics use only one kind of strategy, while others combine two or several strategies.

A variety of heuristic methods has been developed for the MAX-CUT, MAX-SAT and their variations. A recent survey on metaheuristics is given in [78]. The MAX-CUT problem can naturally be implemented as an unconstrained binary quadratic programming problem (UBQP); for survey on the heuristics for UBQP the reader is referred to [27].

We will now review some of the heuristics implemented and applied for problems that are the main interest of this thesis: the MAX-DI-CUT with source and sink, and the MAX- k -SAT problem with $k = 2$, $k = 3$ and $k = 4$. Since the MAX- k -SAT, $k \geq 2$, is the restricted version of the general problem MAX-SAT, obviously, the heuristics developed for the MAX-SAT problem can be applied to. But, these heuristics could perform worse than the heuristics developed specifically for the variations – the MAX- k -SAT, $k \geq 2$. In the unweighted version of MAX-SAT problem the algorithms developed for satisfiability (SAT) problem can be applied straightforwardly by choosing a solution that satisfies most of the clauses. It is not clear whether the heuristics with a good performance for the SAT perform equally good for the MAX-SAT [149].

GRASP

Greedy randomized adaptive search procedure (GRASP) was proposed by Feo and Resende [54, 55], and its application to various optimization problems has been studied, including the MAX-CUT [56] and the WEIGHTED MAX-SAT [122, 131]. GRASP is a metaheuristic

that applies local search on start solutions generated by a greedy randomized construction procedure. In each GRASP iteration a start solution for local search is iteratively constructed. At each step of construction procedure, a set of candidate elements C is constructed, which is called a *candidate list* (CL) and contains all elements that can be added to extend a partial solution. A greedy function $g : C \rightarrow R$ measures the incremental value for the objective function by adding an element $e \in C$ to the current partial solution. In this step, GRASP restricts the set of candidate elements according to the greedy function, then the element to be added into the partial solution is randomly determined from the restricted candidate list (RCL); or it could restrict the candidate list randomly, then the element to be added is determined greedily. The candidate elements $\forall e \in C$ are sorted according to their greedy function values $g(e)$. In a cardinality based RCL construction, the top k elements are chosen to RCL. In the value based construction, RCL is a set $\{e \in C : g_{min} \leq g(e) \leq g_{min} + \alpha(g_{max} - g_{min})\}$, where g_{min} and g_{max} are the lowest and the highest values of g through C , respectively, and $0 \leq \alpha \leq 1$ is a parameter. Often a random value is assigned to α , because it is not entirely trivial to determine the best value for α .

Pardalos, Pitsoulis, and Resende [131] implemented and tested the GRASP for the WEIGHTED MAX-SAT. They generated test instances from the DIMACS SAT instances¹ by associating the integral weights $[1, \dots, 1000]$ to SAT clauses independently and randomly. These test instances² are used by other authors as well to test their algorithms. In [131] the test instances are optimally solved by CPLEX (problem is implemented as MIP), and the solutions found by GRASP are compared to the optimal solutions. GRASP solved in its 10,000 iterations 3 of 44 test instances optimally and found near-optimal solutions (at least 99.86 p.c. of optimal value) for all tested instances.

VNS

The variable neighborhood search (VNS) has been developed by Hansen and Mladenović [110]. This is a multi-start metaheuristic that systematically changes the neighborhood structure within a search. The basic VNS method combines deterministic and stochastic changes of neighborhood. In each iteration of VNS, the start solution for local search is randomly generated from the neighborhood of the current solution and the neighborhood structure is changed iteratively. For some $k_{max} \in \mathbb{N}$ let us denote with N_k , $k = 1, \dots, k_{max}$,

¹<ftp://dimacs.rutgers.edu/pub/challenge/satisfiability/benchmarks/cnf/>

²<http://www.research.att.com/mgcr/data/maxsat.tar.gz>

the finite set of pre-defined neighborhood structures and with $N_k(x)$, $k = 1, \dots, k_{max}$, the set of neighbors in the k -th neighborhood of x . Starting with an initial solution x , VNS randomly explores in its each iteration the k -th neighborhood $N_k(x)$ systematically, $k = 1, \dots, k_{max}$: start k with 1 and stop when k reaches k_{max} , a start solution for local search is generated from the k -th neighborhood of x at random, and an obtained solution through local search updates x and move to the first neighborhood if some given condition is satisfied, otherwise explore the next neighborhood. This step is repeated by changing the pre-selected neighborhood structures N_k , $k = 1, \dots, k_{max}$.

Let f denote the objective function of given optimization problem (maximization). Algorithm 1 shows the pseudocode of the basic VNS algorithm.

Algorithm 1 Basic VNS

```

procedure BasicVNS( $x, k_{max}$ )
  repeat
     $k \leftarrow 1$ 
    while  $k < k_{max}$  do
      Generate a point  $x'$  at random from the  $k$ -th neighborhood of  $x$ :  $x' \in N_k(x)$ 
       $x'' \leftarrow \text{LocalSearch}(x')$ 
      if  $f(x) < f(x'')$  then
         $x \leftarrow x'', k \leftarrow 1$ 
      else
         $k \leftarrow k + 1$ 
      end if
    end while
  until stopping condition is met
  return  $x$ ;

```

The basic VNS is useful for the approximation of many combinatorial and global optimization problems, but it remains to be difficult and takes long computation time for very large instances. In literatures there are some extensions and intensification of VNS studied. We briefly describe some of them in following.

The Variable Neighborhood Decomposition Search (VNDS) [88] method extends the local search phase of basic VNS. Instead of applying local search to k -th neighbor x' of the current solution x in whole solution space, the VNDS applies local search on a subspace of the solution space, specifically only to the components where x and its neighbor x' are different.

The Skewed VNS (SVNS) [87], an another extension, addresses the problem of exploring

solutions far from the current solution. The SVNS differs from basic VNS at the decision phase: only if the local optimum x'' that is found by the local search starting from the neighbor of current solution x lies far enough from x , then the local optima x'' is taken as a next candidate; SVNS keeps the best objective value that is so far seen for output. A good choice of the requesting distance could be made through some learning process.

There are several ways for parallelizing VNS that have been proposed in [36, 64]. In [64] it was shown that an approach which assigns dissimilar neighborhoods to each processor and interrupts their work as soon as an improvement occurs, gives very good results.

An adaptation of VNS to the MAX-CUT [56] yielded good solutions in shorter time compared to GRASP. The SVNS approach is successfully applied to the weighted MAX-SAT problem [87], and reported its efficiency in large instances where it outperformed the GRASP and tabu search heuristics. Moreover, the adaptation of VNS to Max-SAT [87] showed that the VNS heuristic is comparable with simple tabu search, and found better solutions than GRASP.

Path-Relinking

The Path-Relinking (PR) was proposed by Glover [70, 71, 72, 77] as an intensification strategy to explore the trajectories that connect elite solutions obtained by tabu search or scatter search. Path-Relinking method can be integrated into metaheuristics. A strategy of PR is that a book keeping the elite solutions (solutions with high quality) found during the search and exploring the paths connecting elite solutions with each other may yield better solutions that were actually not seen during the search. Indeed, the solutions with high quality can have the attributes of optimal solutions. The paths connecting two elite solutions could be simultaneously explored from both ends and usually one considers the path of the shortest Hamming distance. An Algorithm 2 shows the pseudocode of PR procedure applied to a pair of solutions (x_s, x_t) , start and target solutions.

Algorithm 2 Path-Relinking

procedure PR(x_s, x_t)

 Apply a local search starting from x_s to x_t allowing only moves that reduce the distance to x_t in a subspace where the components of x_s and x_t differ.

return a local optimum found by local search

The procedure starts by computing the symmetric difference between the start and target solutions, the set of components where x_s and x_t are different. Then by fixing the equal

components of both solutions, it applies local search in a sub-space of the whole solution space. In local search it allows only the moves that reduce the distance from the start solution to the target solution. The best solution found in the exploration is returned. Resende and Ribeiro [130] have implemented several alternative path-relinking strategies:

- periodical-relinking: path-relinking is applied only periodically in the search;
- forward-relinking: the worst of given two solutions, x_s and x_t , set as start solution and other as target solution;
- backward-relinking: the best of given two solutions, x_s and x_t , set as start solution and other as target solution.
- back- and forward-relinking: combining both directions, applying local search starting from x_s , as well as x_t ;
- mixed-relinking: combining both directions, applying local search starting simultaneously from x_s and from x_t , and exploring the paths that meet each other.
- randomized-relinking: applying local search that moves into solutions randomly chosen from a list of candidate solutions.
- truncated-relinking: applying local search that explores only the part of paths connecting x_s and x_t .

These alternatives have both positive and negative impacts to the quality of the solution and the running time of the procedure. Choosing the best of the given two solutions as the start solution usually generates the best quality solution, because the neighbors of the starting solution are taken into consideration more than the neighborhood of the target solution.

Hybrids

Festa et al. [57] proposed hybrid heuristics that are derived from GRASP, VNS and path-relinking, and compared them with each other and with GRASP. In [57] the path-relinking is able to improve each of GRASP and VNS in hybrid implementation, specifically, GRASP with PR and VNS with PR, as well as GRASP with VNS and PR (in the local search

phase). GRASP with path-relinking is the fastest among the implemented heuristics [57] to find the solutions with values at least as good as the target-values (suboptima).

In the hybrid implementation of GRASP the path-relinking is performed after the local search phase. The set of elite solutions so far found in the search is kept in a list, and the path-relinking performed with local optimum as the start solution and randomly chosen elite solution from the list as the target solution. More precisely, the randomized-relinking was used. The hybrid implementation of VNS with the path-relinking is similar to of GRASP, the path-relinking implemented as an intensification phase after each local search phase. In the hybridization of GRASP with VNS, in the second phase of GRASP the local search is substituted by the VNS – a start solution generated in the first phase of GRASP is directly given into the VNS procedure. The hybrid GRASP with VNS and PR was implemented in a way that, the hybrid VNS with PR was performed in the local search phase of GRASP.

Festa et al. [57] also reported that these hybrid approaches achieved better solutions, compared to the *rank-2 relaxation* heuristic proposed by Burer, Monteiro and Zhang [31], but in a longer computation time.

Tabu Search

A Tabu Search (TS) is a metaheuristic developed by Glover [76]. TS guides the search to escape from local optima. A basic idea of TS is adding short-term memory that improves the ability to locate optimal solutions. Revisiting previously or recently visited solutions is prevented, and the operations that would do so are labeled as being *tabu*. A simple memory is a *tabu list* that keeps the track of the recently visited solutions. In each iteration of the search, a neighborhood structure of the incumbent is modified such that the search escapes from the local optima. The modifications of neighborhood structure can be done by various ways. For example, the neighbors of the incumbent, which are in the tabu list, are prohibited, so these neighbors are excluded from the search. As solutions can be prohibited, the certain attributes of the solutions can be stored in the memory, to avoid doing this change, or not changing (to keep) these attributes.

The simple Tabu Search can be implemented as shown in Algorithm 3. In each iteration of the search, first, some selection-function decides which of the solutions in the tabu list (*TL*) can be included into or excluded from the neighborhood of the incumbent solution. This selection-function is denoted as *Allow* in the pseudo-code. Then a local search is performed into this extended (or modified) neighborhood of the incumbent. An obtained

solution is added into the tabu list. Since the search iterations may be large, holding all the solutions found during the search in the tabu list would be inefficient. In the Algorithm 3 the size of TL is limited by an l . The versions and extensions of TS vary by handling of the length of the tabu list, how long the solutions would be stored in memory, and by the selection function.

Algorithm 3 Tabu-Search

```

 $TL \leftarrow \emptyset$ 
Construct a random solution  $x_0$ 
repeat
   $N^*(x) \leftarrow Allow(N(x) \cup TL)$ 
   $x' \leftarrow LocalSearch(x, N^*(x))$ 
  if  $|TL| \geq l$  then
     $TL \leftarrow TL \setminus \{\text{first element in } TL\}$ 
  end if
   $TL \leftarrow TL \cup x'$ 
  if  $f(x') > f(x)$  then
     $x \leftarrow x'$ 
  end if
until

```

Glover [73] proposed methods that use the statically- and dynamically-sized memory structures for tracking tabu operations. Taillard [151] proposed the Robust Tabu Search (RoTS), which introduced a dynamic randomly-sized short-term memory design. Battiti and Tecchiolli [24] developed the Reactive Tabu Search (ReTS) that is based on the dynamic size of its short-term memory on runtime characteristics of the algorithm, and also utilizes a form of long-term memory that helps to prevent the search from stagnating. Many other Tabu Search variations have been developed that combine the various forms of dynamically-sized short-term memory and long-term memory [74, 76], but the RoTS and ReTS are among the most successful and popular. Other approaches are developed through an experimentation with features such as socialization and competition [43] or like the EE-TS [108], the integration of evolutionary operators useful for multimodal optimization.

Many applications and adaptations of tabu search to variety of optimization problems have been studied [76].

The Reactive Tabu Search

The Reactive Tabu Search algorithm (Reactive-TS) [22] is a tabu search that uses a parameter, that determines for how many iterations the prohibited (during the search) solutions should be kept in the memory (tabu list), varies dynamically during the search. This parameter changes depending on solutions' appearance in the neighborhood of the current incumbent solution. An empirical study [22] shows that Reactive-TS performs well for the MAX-SAT problem. Reactive-TS was successfully applied to various optimization problems and showed its competitiveness with other metaheuristics [21].

Iterated Robust Tabu Search

Smyth, Hoos and T. Stützle [149] have proposed and studied the Iterated Robust Tabu Search (IRoTS) for the weighted MAX-SAT problem. IRoTS is stochastic, adaptive and memory based multi-start heuristic method. The local search and the start solution generation of IRoTS are both based on the adaptation of RoTS [151] to the weighted MAX-SAT problem. Therefore, it derives the name of this heuristic. IRoTS starts with initializing each variable with the random value 0 or 1 independently (with equal probability) and obtains an initial solution. Then the local search starts from the initial solution. Each iteration of IRoTS consists of two phases, a perturbation phase that helps the search to escape from the local optima and the local search phase. The results of perturbation phase are the start solutions for local search. The perturbation phase is initialized by the best solution found in the search or by the result of the previous local search (with a certain probability); and the start solution for local search is generated by performing the fixed number of RoTS steps. In the local search the RoTS steps are performed until no improvement in the incumbent solution has been achieved for a certain number of iterations.

In experimental study of Smyth, Hoos and T. Stützle [149] IRoTS was tested to the instances, which have been previously proposed in the literature, as well as to the random instances. IRoTS was compared with GLS [109] and Yagiura and Ibaraki's algorithm for MAX-SAT [162], which were outperformed by the IRoTS. The experimental study shows that IRoTS performs faster than GLS to the random instances and to the instances with high ratio of unsatisfiable and satisfiable clauses (over constrained instances). IRoTS and GLS perform similar to the instances for which the GLS was reported to be the best algorithm.

DLM

Discrete Lagrangian Method (DLM) [158] is one of the methods that extend the traditional Lagrangian method for solving continuous optimization problems [105] to the discrete constrained optimization problems. DLM is another strategy to escape from local optima in the search. When the search reaches a local optimum, the *Lagrange multipliers* give the force to move out of local optimum to the direction provided by Lagrange multipliers. This attribute gives a possibility to produce the continuous trajectory in the search, compared to the simple local search that forces to restart when local optimum is reached. Wah and Shang [158] proposed DLM for the MAX-SAT problem. In their computational study the DLM was compared with GRASP ([131]) on DIMACS SAT benchmark instances ¹. The study suggests that DLM to be generally 1 to 3 orders of magnitude faster than GRASP, but for some classes of test problems DLM to be worse than GRASP. These benchmark problems have 100 variables and clauses vary from 800 to 900.

GLS

Guided Local Search (GLS) [157] is another history based metaheuristic, which helps the local search to escape from local optima. GLS uses the penalties on the features of solutions. In each of its iteration GLS performs the local search procedure starting from the previously found local optimum (instead of starting from a random solution) by dynamically changing an augmented objective function. This allows to move out of local optima. Local search is profited by problem specification and search history. The augmented objective function incorporates the original objective with feature penalties. In each iteration, the penalties are modified, and so does augmented objective function. The features of solutions and penalty modification should be suited into underlying problem. For example, in case of MAX-SAT, the penalty of the feature (clause) is increased, for the one that has the largest ratio of increment in the objective function and its penalty.

GLS has been successfully applied on many optimization problems. Mills and Tsang [109] implemented the GLS for SAT and WEIGHTED-MAX-SAT problems. In their computational study the GLS has been tested and compared with GRASP and DLM on 44 WEIGHTED-MAX-SAT benchmark instances. Empirical results show that GLS performs very well: it has found the optimal solutions for all tested problems in the best of 20 individual runs.

¹[ftp://dimacs.rutgers.edu/pub/challenge/satisfiability/benchmarks/cnf/](http://dimacs.rutgers.edu/pub/challenge/satisfiability/benchmarks/cnf/)

GRASP [131] is outperformed by GLS, while DLM [158] is comparable with GLS, where DLM found the optimal solutions for 39 instances (88% of all instances) in the best of 20 individual runs.

Genetic Algorithms

The hybridizations of genetic algorithms and metaheuristics showed good results to the MAX-CUT problem. For example, one of the best is a memetic algorithm proposed by Kim, Kim and Moon [99], that is a hybridization of a genetic algorithm with a local search procedure analogous to the Fiduccia-Mattheyses algorithm [58]. Their hybrid algorithm yields better solutions than a hybridization of the Goemans-Williamson's approximation algorithm with simulated-annealing as proposed by Homer and Peinado [94] (who implemented the SDP-relaxation as a constrained nonlinear programming problem and stated that better cut values were found than their implementation of the Goemans-Williamson's algorithm, in less time).

Cross-Entropy Method

Cross-Entropy (CE) method [42] is an iteratively adaptive and randomized approach based on an associated stochastic network (which is a transformation of the deterministic network of the underlying problem). Each iteration of the CE method on the associated stochastic network has two phases, (a) generate a random data sample according to a specified mechanism, and (b) update the parameters of the random mechanism based on the data generated in phase (a) to produce a better sample in the next iteration. Rubinstein [133] proposed a CE algorithm that performed well for the MAX-CUT problem.

3.5. Hybrid LP-based Approaches

In recent years various methods have been proposed to solve hard combinatorial optimization problems combining exact and metaheuristic algorithms. These methods are shown to benefit from the advances of both approaches, exact and metaheuristic algorithms. Dumitrescu and Stützle [51, 50] and Puchinger and Raidl [129] surveyed such hybridized methods, and classified them. A latter study identified two main classes: *collaborative* and *integrative*. In *collaborative combinations* of exact and metaheuristic algorithms, the algorithms are not part of each other, and may be executed subsequentially or in parallel. In

integrative combinations, one algorithm is embedded into another algorithm. According to this classification, our technique fits into the collaborative combination class, i.e., the two-stage algorithm: linear programming first, and then local search to the optimal solution of the LP.

The special combinations, the hybridizations of LP and the metaheuristic, are proposed for several combinatorial optimization problems. In these combinations the fractional solutions of LP are rounded into integral solutions by some rounding techniques, and the integral solutions guide heuristic procedures.

In contrast, in our approach the underlying LP generates directly a feasible solution of the primary problem, which guides the local search. We briefly describe several hybrid LP-based heuristic approaches.

French and Wilson [61] proposed an LP-based heuristic algorithm for generalized assignment problem (GAP) with special ordered sets of type two [26], the extension of GAP. The algorithm is an extensive adaptation of heuristic algorithm for GAP. Their heuristic method consists of two phases. In the first phase the series of LP-relaxations is solved iteratively within heuristic procedure. A feasible solution obtained in the first phase is improved by a local search in the second phase of the algorithm. The rational valued variables of the LP-solution, which satisfy certain criterion are fixed at “0”, and this information is used to generate the LP in the next iteration.

Klose [100] proposed an LP-based heuristic for the two-stage capacitated facility location problem. An LP formulation, which is a relaxation of the original problem, is iteratively refined using known valid inequalities and facets for various relaxations of the problem. The method solves a series of LP-relaxations. At each iteration, a feasible solution is obtained from a fractional solution of LP by applying (simple) heuristic.

Sridhar et al. [150] proposed a heuristic method for a capacitated network design problem. An aim of the problem is to find a feasible solution, a topology of the network. Again, they solve an LP relaxation of an MIP formulation of the initial problem, and the fractional solution of LP is used by some heuristic to build a feasible solution. The method uses subsequentially several heuristics to improve the feasible solution.

Umetani et al. [154] proposed a local search approach based on LP for a special version of the one dimensional cutting stock problem, which they have stated as *pattern restricted problem*. In the algorithm, a certain LP relaxation is solved. The optimal solution of this LP is rounded to get a feasible solution of primary problem.

Alvarez-Valdes et al. [4] developed an LP-based heuristic method for solving a two-dimensional cutting stock problem. The algorithm is based on the Gilmore and Gomory column generation scheme [67, 68] and consists of two steps, an iterating step and a rounding step. In each iteration of the first step, a certain LP relaxation of the problem is solved and the LP-solution is used to generate the subproblems (the subproblems can be solved exactly, or by applying heuristic methods). Depending on the solutions of subproblems, the iterating step continues, or stops. If the iterating procedure continues, the solutions of subproblems are used to generate the LP relaxation of the primary problem in the next iteration. In the second step, the feasible solution of the primary problem is obtained by rounding the LP-solution.

Vasquez and Hao proposed LP based heuristic approach for the 0-1 multidimensional knapsack problem [155]. The algorithm involves a subsequent run of the LP solver and the heuristic method. The algorithm solves a series of certain LP relaxations of the primary problem. The fractional solutions of the LPs are rounded by some rounding scheme. The obtained integral solution is taken as a start solution for tabu search.

Chapter 4

Local Search Starting From an LP Solution

In this chapter we introduce the LP-based technique that generates the start solutions for local search to the MAX-DI-CUT problem with source and sink, the MAX- k -SAT problem, where $k \geq 2$, and the LONGEST DIRECTED PATH problem with source and sink. Clearly, the proposed method applies to the MAX-2-SAT, MAX-3-SAT and MAX-4-SAT problems, since these are the variations of general problem where $k = 2$, $k = 3$ and $k = 4$, respectively.

4.1. Introduction

The main goal of our technique is to solve the given problem fast. The success of the simple local search crucially depends on the start solutions. Consequently, providing the local search with good start solutions that are near to optimal solutions could lead to a fast and efficient algorithm. We present an LP based technique that generates start solutions of apparently high quality. This LP based technique might apply to a variety of optimization problems but adaptation to a given problem does not seem to be entirely trivial. Therefore, we must particularly consider the underlying problem to develop this technique.

It is a requirement for our method to work that there be an LP such that the optimal basis solutions to the LP be feasible solutions to the original problem. This does not mean that this LP describes the original problem. We also demonstrate where it is promising to look for such an LP: take a quadratic programming (QP) formulation and try to develop an LP that meets this requirement.

In the following sections we present our LP based technique for the MAX-DI-CUT problem with source and sink, the MAX- k -SAT problem, where $k \geq 2$, and the LONGEST DIRECTED

PATH problem with source and sink. We consider the nonlinear formulations of the MAX-DI-CUT and MAX- k -SAT, $k \geq 2$, problems and apply small modifications to transform the non-linear ingredients into linear ones. In each case, we define the modification such that the vertices of the resulting LP are integral, and that the simplex method will not end up at infinity (although the polyhedron itself will be unbounded in general). In case of the MAX-DI-CUT we substitute the quadratic objective function by linear function, and in case of the MAX- k -SAT we substitute the quadratic constraints by linear constraints to get the desired LPs. In case of the LONGEST DIRECTED PATH we obtain the required LP by alternative way – incorporating flow consistency inequalities. We use the LONGEST DIRECTED PATH problem as an example to corroborate that (I) our technique is not suitable for arbitrary problem, but, (II) our technique achieves near optimal solutions to the MAX-DI-CUT problem with source and sink and the variations of the MAX- k -SAT problem, where $k \in \{2, 3, 4\}$. The two-phase algorithm that results from our LP-based local search technique is as follows:

- 1) *solve the corresponding LP;*
- 2) *run the local search starting at the LP solution.*

4.2. Directed Max-Cut with Source and Sink

Problem definition: Recall the definition of the Section 1.1. An input consists of a directed graph $G = (V, A)$, a nonnegative weight $c_a \in \mathbb{R}_0^+$ for each arc $a \in A$, and two nodes, $s, t \in V$ (source and sink). A feasible solution – an (s, t) -cut – is a partition of V into two subsets, S and T , such that $s \in S$ and $t \in T$. The objective function to be maximized is the weight of the (s, t) -cut:

$$C(S, T) = \sum_{\substack{(v,w) \in A \\ v \in S, w \in T}} c_{v,w}.$$

Local search: We generate a start solution as described below. The local-search procedure is based on a simple neighborhood structure: two (s, t) -cuts, (S_1, T_1) and (S_2, T_2) , are neighbored if, and only if, they differ on exactly one node. More formally, this means $|S_1 \setminus S_2| + |S_2 \setminus S_1| = 1$.

Generating a start solution: An (s, t) -cut, (S, T) , may be identified with its characteristic vector $x = (x_1, \dots, x_n) \in \{0, 1\}^n$, $n = |V|$: $x_v = 1$ if, and only if, $v \in S$. For each

arc $a = (v, w) \in A$, we define a variable $y_a \in \{0, 1\}$ such that $y_a = 1$ if, and only if, $v \in S$ and $w \in T$.

We will solve the following LP using the simplex algorithm and take the result as the start solution:

$$\begin{aligned}
 \sum_{a \in A} c_a \cdot y_a &\longrightarrow \max \\
 y_a &\leq x_v - x_w \quad \forall a = (v, w) \in A \\
 x_s - x_t &\geq 1 \\
 x_v &\leq 1 \quad \forall v \in V \\
 x_v &\geq 0 \quad \forall v \in V.
 \end{aligned} \tag{4.1}$$

Proposition 4.2.1 *The vector $x = (x_1, \dots, x_n)$ in the solution of LP (4.1) yields an (s, t) -cut.*

Proof: For correctness, we have to argue that (i) the vertices of the polyhedron have integral x -values, and (ii) for no start basis does (phase II of) the simplex algorithm end up at infinity. We confidently assume that degeneracies and numerical inaccuracies are handled appropriately by the simplex implementation.

Integrality of the x -values of the vertices follows directly from the following theorem, which has been proved by Garg and Vazirani [66]. In fact, the sign change in the first set of inequalities does not have any impact on the integrality of the x -values.

Theorem 4.2.2 [66] *The vertices of the following polyhedron are exactly the (s, t) -cuts:*

$$\begin{aligned}
 y_a &\geq x_v - x_w \quad \forall a = (v, w) \in A \\
 x_s - x_t &\geq 1 \\
 x_v &\leq 1 \quad \forall v \in V \\
 x_v &\geq 0 \quad \forall v \in V \\
 y_a &\geq 0 \quad \forall a \in A.
 \end{aligned} \tag{4.2}$$

It remains to argue that the simplex algorithm cannot end up at infinity. Consider a non-degenerate basis exchange step. By definition this step increases the objective function value. So, the value of at least one of the variables y_a is increased in such a step. Since $y_a \leq x_v - x_w \leq 1$ for all $a = (v, w) \in A$, this exchange step is finite. In summary, correctness is proved. \square

Note that (4.1) is *not* a correct model of the MAX-DI-CUT problem. However, there is indeed a strong relation: it is easy to see that

$$\frac{1}{2} \sum_{a \in A} c_a (y_a^2 + y_a) \longrightarrow \max$$

subject to the side constraints of (4.1) is a correct model. So, (4.1) is the linear version of a certain quadratic-programming formulation of the MAX-DI-CUT problem.

4.3. Max- k -SAT

Problem definition: An input consists of positive integral numbers, m , n and $k \geq 2$, and m nonnegative weighted clauses on n 0 – 1 variables, each clause consisting of at most k literals. The variables will be denoted by x_1, \dots, x_n , and the clauses by z_1, \dots, z_m . As usual, a literal is either x_i or \bar{x}_i (that is, x_i negated). For each clause z_j , $j \in \{1, \dots, m\}$, the input contains a nonnegative weight $w_j \in \mathbb{R}_0^+$.

A feasible solution assigns a value 0 or 1 to each variable x_i , $i \in \{1, \dots, n\}$. The objective to be maximized is the sum of the weights of the clauses that are satisfied by this assignment of 0 – 1 values.

Local search: Below we will describe how to generate the start solution. We use the following neighborhood structure: two truth assignments, (x_1^1, \dots, x_n^1) and (x_1^2, \dots, x_n^2) , are neighbored if, and only if, they differ on only one variable. Formally, this means $\sum_{i=1}^n |x_i^1 - x_i^2| = 1$.

Generating a start solution: For each $i \in \{1, \dots, n\}$, we define an additional variable x_{n+i} and require $x_{n+i} = 1 - x_i$. Then each clause z_j can be written as $z_j = x_{i_1} \vee \dots \vee x_{i_k}$ for some $i_1, \dots, i_k \in \{1, \dots, 2n\}$. We identify a clause $x_{i_1} \vee \dots \vee x_{i_k}$ with the indices of its literals (i_1, \dots, i_k) . Let Z denote the input set of all clauses. For each clause $(i_1, \dots, i_k) \in Z$, we define a variable $y_{i_1, \dots, i_k} \in \{0, 1\}$ such that $y_{i_1, \dots, i_k} = 0$ if, and only if, the clause (i_1, \dots, i_k) does not satisfy. We will solve the following LP using the simplex method and

take the solution as the start solution:

$$\begin{aligned}
& \sum_{(i_1, \dots, i_k) \in Z} w_{i_1, \dots, i_k} \cdot y_{i_1, \dots, i_k} \longrightarrow \max \\
& y_{i_1, \dots, i_k} \leq x_{i_1} + \dots + x_{i_k} & \forall (i_1, \dots, i_k) \in Z \\
& x_i + x_{n+i} \leq 1 & \forall i \in \{1, \dots, n\} \\
& x_i \leq 1 & \forall i \in \{1, \dots, 2n\} \\
& x_i \geq 0 & \forall i \in \{1, \dots, 2n\}.
\end{aligned} \tag{4.3}$$

Proposition 4.3.1 *The vector $x = (x_1, \dots, x_n)$ in an each solution of LP (4.3) yields the truth assignment.*

Proof: We have to show that (i) the vertices of the polyhedron have integral x -values, and (ii) for no start basis does (phase II of) the simplex algorithm end up at infinity.

The value of variable y_{i_1, \dots, i_k} , $\forall (i_1, \dots, i_k) \in Z$, is defined by only one inequality in the side constraints. Hence, the projection of the constraint polyhedron onto the $x = (x_1, \dots, x_n)$ -space yields an n -dimensional hypercube.

The proof that the simplex method will not end up at infinity is perfectly analogous to the directed MAX-CUT problem (Section 4.2). So, the simplex method will find a truth assignment at all odds. \square

This LP is *not* a correct model of the MAX- k -SAT problem. However, it can be transformed into a correct model. Let $F_l(x_{i_1}, \dots, x_{i_k})$ denote the set of all clauses with length $l \leq k$ that are constructed of literals x_{i_1}, \dots, x_{i_k} . Then just replace every linear side constraint “ $y_{i_1, \dots, i_k} \leq x_{i_1} + \dots + x_{i_k}$ ” by the non-linear constraint

$$y_{i_1, \dots, i_k} \leq \sum_{l=1}^k (-1)^{l-1} \left(\sum_{(x_{h_1}, \dots, x_{h_l}) \in F_j(x_{i_1}, \dots, x_{i_k})} x_{h_1} \dots x_{h_l} \right).$$

So again, the LP (4.3) is the linear variant of a certain nonlinear-programming formulation of the MAX- k -SAT, $k \geq 2$.

4.4. The Longest Path with Source and Sink

Recall the definition of the restricted version of the longest path problem (Section 1.1), in which an objective is to find the longest path between source and sink in a directed

and weighted graph. An input consists of a directed graph $G = (V, A)$, a positive cost (or length) $c_a \in \mathbb{R}^+$ on each arc $a \in A$, and two nodes, $s, t \in V$ (source and sink). Note that the source has no entering arc and the sink has no leaving arc, $\delta_s^{in} = \emptyset$ and $\delta_t^{out} = \emptyset$. An $s-t$ path is a sequence of distinct vertices $P = \{v_1, v_2, \dots, v_k\}$ such that for any $1 \leq i \leq k-1$, (v_i, v_{i+1}) is an arc, $(v_i, v_{i+1}) \in A$, and $v_1 = s$ and $v_k = t$. The length of the $s-t$ path P is the sum of lengths of arcs that are on the path

$$C(P) := \sum_{i=1}^{k-1} c_{v_i v_{i+1}}.$$

An objective is to find an $s-t$ path with maximum length.

Local search: We use the depth first search (DFS) to move to the neighbored path. A pseudocode of the local search for the longest path with source and sink (**LocalSearchLP**) is shown in Algorithm 4. Let P denote the incumbent solution. Algorithm starts by setting a given start $s-t$ path P_0 as a current incumbent $P = \{v_1, \dots, v_k\}$. We run DFS starting at each node of the incumbent path, except the tale $-t$ of the path. As soon as DFS finds longer path, we move to such a path immediately, and update the incumbent. We repeat this procedure until no improvement happens. Note that $s = v_1$ and $t = v_k$.

In the DFS, the successor nodes are selected at random, in other words, we randomly move to the successor. In this respect the local search is randomized.

Algorithm 4 Local Search

```

procedure LocalSearchLP( $P_0$ )
  set  $P \leftarrow P_0$ ; let  $P = \{v_1, \dots, v_k\}$ ;
  repeat
     $i \leftarrow |P|$ ;  $v_i = t$ ;
    while  $v_i \neq s$  do
       $i \leftarrow (i - 1)$ 
      start DFS at  $(v_i)$ ;
      if DFS finds longer path then
        update  $P$ ;
      end if
    end while
  until no improvement occurs;
  return  $P$ ;

```

Generating a start solution: We introduce an LP, whose optimal basis solutions contain $s - t$ paths. An $s - t$ path P may be identified with the following variables. For each node $v \in V$, we define a variable $x_v \in \{0, 1\}$ such that $x_v = 1$ if, and only if, $v \in P$. Moreover, for each arc $a = (u, w) \in A$ let variable $y_a \in \{0, 1\}$ denote whether the $s - t$ path traverses over arc a : $y_a = 1$ if, and only if $\exists i, 1 \leq i \leq k - 1$ such that $u = v_i$ and $w = v_{i+1}$.

The following LP is derived from flow conservation inequalities and its solutions may contain isolated cycles.

$$\begin{aligned}
& \sum_{a \in A} c_a \cdot y_a \longrightarrow \max \\
& \sum_{(u,w) \in A} y_{uw} = x_u \quad \forall u \in V \setminus \{t\} \\
& x_t = 1 \\
& x_s = 1 \\
& \sum_{(w,u) \in A} y_{wu} = x_u \quad \forall u \in V \setminus \{s\} \\
& 0 \leq x_u \leq 1 \quad \forall u \in V \\
& 0 \leq y_a \leq 1 \quad \forall a \in A
\end{aligned} \tag{4.4}$$

Proposition 4.4.1 *The optimal basis solutions of LP (4.4) are integral.*

Proof: A constraint matrix associating the LP (4.4) is totally-unimodular. Therefore, the optimal basis solutions of (4.4) are integral (see [138]). \square

LP (4.4) does not describe the LONGEST DIRECTED PATH problem with source and sink. Indeed, the optimal basis solutions may contain isolated cycles if there exists at least one. However, the optimal basis solutions contain $s - t$ paths within, if there exists at least one. If LP is infeasible then the primary LONGEST DIRECTED PATH problem is also infeasible.

We solve LP (4.4), then extract the enclosed $s - t$ path from the optimal solution. The obtained $s - t$ path is a start solution for the local search.

Chapter 5

Computational Study

In this chapter, we deal with the experimental evaluation. We describe the computational environment and the experimental design, and report the computational results of each experiment in-depth. We test our methods to instances of the MAX-DI-CUT problem, the three variations of the MAX- k -SAT problem with $k = 2$, $k = 3$ and $k = 4$, and the LONGEST DIRECTED PATH.

5.1. Experimental Setup

All experiments were performed on a PC with an AMD Athlon 1,700 MHz single processor and 1Gb of memory. The linear programs were solved by a CPLEX solver of ILOG OPL Studio 3.5.1 [134] and CPLEX shared library using the ILOG Concert technology [135] on the same machine.

We compared our technique to quite a simple *reference technique* which is a multi-start heuristic: a repeated local search from random start solutions. For this local search, we used exactly the same neighborhood structure as for our own technique. For convenience we indicated our technique as **LPcut**, and the reference technique as **REF**. Let f denote the objective function of the given problem. Algorithm 5 shows the pseudocode of the reference technique.

To demonstrate the stability of our technique, we tested our technique to a variety of instance classes of four problems, the MAX-DI-CUT with source and sink, the MAX-2-SAT, MAX-3-SAT, MAX-4-SAT and LONGEST DIRECTED PATH with source and sink (Tables 5.1, 5.3.2, 5.7, 5.10 and 5.12). Tables 5.2, 5.5, 5.8, 5.11 and 5.13 summarize the results of the

Algorithm 5 Reference Strategy

```

procedure REF( $k_{max}$ )
  for  $k = 1, \dots, k_{max}$  do
    Generate a random solution  $x$ 
     $x' \leftarrow \text{LocalSearch}(x)$ 
    if  $k = 1$  then
       $x^* \leftarrow x'$ 
    else if  $f(x^*) < f(x')$  then
       $x^* \leftarrow x'$ 
    end if
  end for
  return  $x^*$ 

```

main part of our computational study. We ran the reference procedure with 1,000 random start solutions ($k_{max} = 1,000$).

We compared the start and final solutions of local search in the LPcut procedure to examine the improvement of the start solution through local search. These comparisons are summarized in the Tables 5.3, 5.6 and 5.9.

Beside that, we ran another experiment (*iteration-to-target-value*) to find out to which extent these 1,000 runs were actually necessary. More specifically, we ran the reference procedure as often as necessary to match or beat our own technique. This is very similar to the time to sub-optimal target value method as described in [1]. The main difference is that we take the number of repeated runs of local search. We denote this procedure as TV and give a pseudocode in Algorithm 6. Supposing a value of the solution of LPcut as a sub-optimal *target-value*, the number of iterations that TV does require can be considered as an iteration-to-target-value.

The procedure TV(target-value) repeatedly performs local search with random start solutions similarly to the procedure REF. But, the difference is that it gets additionally the suboptimal target value $\tilde{f} = f(\tilde{x})$ as an input. The procedure iterates until it finds a solution with value at least as good as \tilde{f} or until the maximum number of iteration q_{max} is reached. On success, it returns the iteration number. Otherwise, a value *false* is returned (unsuccessful).

Additionally, we compared our empirical results with the theoretical bounds that are available for a certain type of the MAX-2-SAT instances.

Algorithm 6 Iteration To Target Value

```

procedure TV( $\tilde{f}, q_{max}$ )
   $k \rightarrow 0$ 
  repeat
     $k \rightarrow k + 1$ 
    Generate a random solution  $x$ 
     $x' \leftarrow \text{LocalSearch}(x)$ 
    if  $\tilde{f} \leq f(x')$  then
      return  $k$ 
    end if
  until  $k = q_{max}$ 
  return false

```

5.2. Directed Maximum Cut with Source and Sink

This section devotes for the experimental results of our technique for the MAX-DI-CUT problem. We discuss the generation of various classes of problem instances and empirical results.

5.2.1. Generation of Test Instances

We used 12 classes of random instances for the experimental study. Table 5.1 summarizes the attributes of instances of each class: the number of instances in that class, the number of nodes (or range), the density (or range), the construction of arc weights, and which generator has been used. All instance graphs have integral arc weights.

The instances of the first and the second classes have been generated by self-written graph generators. We describe these generators. The output of the generator is a directed graph $G = (V, A)$ with weight function c . The first class contains random graphs generated as follows. For a given number of nodes n and the positive integers k and ℓ , the generator constructs a set of arcs A at random such that:

- Without loss of generality, let $s = 1$ and $t = n$.
- For each node $v \in V \setminus \{t\}$ the outdegree d_v^{out} – the number of arcs that leave the node v , and for each node $v \in V \setminus \{s\}$ the indegree d_v^{in} – the number of arcs that enter the node v – are set independently and randomly from a uniform distribution of a range $[1, \dots, k]$: $d_v^{in}, d_v^{out} \in [1, \dots, k]$;

class ID	number of instances	number of nodes	density	arc weight	generator type
1	200	[119–2498]	[0.21–4.42]	uniform distr. of $[1, \dots, 5]$	self-written ^a
2	500	[602–1600]	5.00	uniform	rudy: <i>simple-random</i>
3	1000	[500–4759]	0.50	uniform	rudy: <i>simple-random</i>
4	500	800	[0.50–15.47]	uniform	rudy: <i>simple-random</i>
5	500	[1000–1500]	[0.78–1.17]	uniform	rudy: <i>almost-planar</i>
6	40	[528–5005]	[0.50–11.40]	uniform	rudy: <i>simplex</i>
7	100	2000	0.59	uniform distr. of $[1, \dots, 100]$	rudy: <i>almost-planar</i>
8	100	800	6.00	uniform distr. of $[1, \dots, 10000]$	rudy: <i>simple-random</i>
9	100	[591–994]	0.70	uniform distr. of $[1, \dots, 10000]$	rudy: <i>simple-random</i>
10	254	[1014–4556]	[0.13–0.61]	uniform distr. of $[1, \dots, 3000]$	self-written ^b
11	54	[800–3000]	[0.13–6.00]	various	rudy: various types ^c
12	100	[2000–4000]	[0.10–0.20]	uniform	rudy: <i>toroidal-grid</i>

^arandom graphs

^bThis class contains “*nearly planar*” graphs generated by a self written graph generator.

^cThis class contains graphs similar to those in a *G-set*.

Table 5.1: Various classes of test instances for the MAX-DI-CUT problem

- A source s has no entering arc and a sink t has no leaving arc, $\delta_s^{in} = \emptyset$ and $\delta_t^{out} = \emptyset$.
- For each node, the adjacent nodes are successively chosen from a uniform distribution of nodes respecting the indegree and the outdegree of nodes;
- For each arc a , the weight c_a is chosen randomly from a uniform distribution of the range $[1, \dots, \ell]$.

We generated 200 instances by setting $k = 6$, $\ell = 5$ and randomly choosing n from the uniform distribution of range $[100, \dots, 2500]$.

The second self-written generator produces “*nearly planar*” graphs. “Nearly planar” means that the graph is embedded in a plane, and removing some (small number) of its

edges results the graph to be planar. For a given number of nodes n and a positive integer $\ell \in \mathbb{Z}^+$, the generator constructs the graph $G = (V, E)$ on the plane as follows:

- Draws some triangle Δ with vertices (v_1, v_2, v_3) ; $V = \{v_1, v_2, v_3\}$, $E = \{v_1v_2, v_2v_3, v_1v_3\}$;
- Iteratively, it drops a random point p within the triangle Δ and connects p with the corners of triangle $\Delta' = (w_1, w_2, w_3)$, in which p is fallen and has the closest corners to p , until the given number of nodes is created; $V = V \cup \{w_1, w_2, w_3\}$, $E = E \cup \{w_1p, w_2p, w_3p\}$;
- Finally, draws a small number of additional edges at random that destroy planarity in general;
- Each edge $e \in E$ has weight c_e chosen from a uniform distribution of the range $[1, \dots, \ell]$ at random.

The second generator generates an undirected weighted graph. To transfer the graph into an instance of the MAX-DI-CUT with source and sink, we set without loss of generality $s = 1$ and $t = n$. Moreover, the orientations of arcs are chosen at random, such that the source has no incoming arc and the sink has no outgoing arc.

For the tenth class of instances, we generated 254 “nearly planar” graphs with nodes varying from 1,000 to 4,556, and the weights of edges are independently chosen from a uniform distribution of the range $[1, \dots, 3000]$ at random.

All other instances have been generated by a public domain graph generator, **rudy**¹. This generator generates various types of graphs, random as well as structured, which are described in an Appendix A. The instances of classes 2 to 6 are all unweighted graphs, i.e., the arcs have uniform weights. The instances of classes 7, 8, 9 and 10 have diverse weights. The instances of 7th class have weights vary from 1 to 100, while the instances of 8th and 9th classes have highly diverse weights, which are chosen from a uniform distribution of the range $[1, \dots, 10000]$. The 11th class contains graphs similar to those in a *G-set*, which has been generated by Helmberg and Rendl [91]. *G-set* has been used to test their algorithm implementing a bundle method for solving SDP and by other authors as well [31, 49, 57]. Some of our test instances significantly differ by weight functions from the graphs of *G-set*, since all of our test instances have positive weights. These graphs are various types of the **rudy**, including *simple-random*, *simplex*, *toroidal-grid* and *almost-planar*.

¹<http://www-user.tu-chemnitz.de/helmberg/semidef.html>

	Comparison of objective values					Comparison of CPU times		
ID	the least	90% \geq	average	the most	percentage	the most	90% \leq	average
(A)	(B)	(C)	(D)	(E)	(F)	(G)	(H)	(I)
1	98.15	99.35	101.9	103.7	78.5	0.0456	0.0053	0.0037
2	99.86	99.95	99.98	100.0	10.0	0.00472	0.00257	0.00183
3	99.28	99.72	99.9	100.9	33.5	0.02168	0.01623	0.00519
4	99.49	99.89	99.96	100.0	20.0	0.0224	0.0182	0.0114
5	99.73	99.86	99.9	100.0	3.0	0.014	0.0032	0.0021
6	99.65	99.84	99.899	100.0	30.0	0.0122	0.013	0.0075
7	99.85	99.88	99.92	99.99	0.0	0.004	0.0025	0.002
8	99.89	99.97	99.98	100.0	28.0	0.0098	0.0088	0.0074
9	99.26	99.62	99.77	99.96	0.0	0.0045	0.0038	0.0027
10	99.72	100.0	100.55	101.27	96.0	0.0047	0.0039	0.0029
11	95.5	98.2	99.6	100.0	2.27	0.0097	0.009	0.004
12	97.1	97.17	97.41	98.38	0.0	0.0011	0.0008	0.0006

Table 5.2: Comparison of our technique with the reference technique for MAX-DI-CUT problem.

5.2.2. Experimental Evaluation

The first experiment consisted of twelve parts, each part considering one class of the test instances. We summarize the computational results that are the comparison of objective values and the comparison of CPU times of the `LPcut` and `REF` in a Table 5.2. The columns of the Table 5.2 describe:

- (A) ID of the experiment;
- (B) – (E) ratios (p.c.) of the objective values found by `LPcut` and by `REF`:
 - (B) the worst,
 - (C) for 90 p.c. of all instances the ratio (p.c.) hits or exceeds this value,
 - (D) the average,
 - (E) the best;
 - (F) the percentage of instances where the outcome of `LPcut` was better than or equal to the outcome of `REF`;
- (G) – (I) ratios of the CPU times needed by `LPcut` and the `REF`:
 - (G) the most;
 - (H) for 90 p.c. of all instances the ratio does not exceed this value;
 - (I) the average.

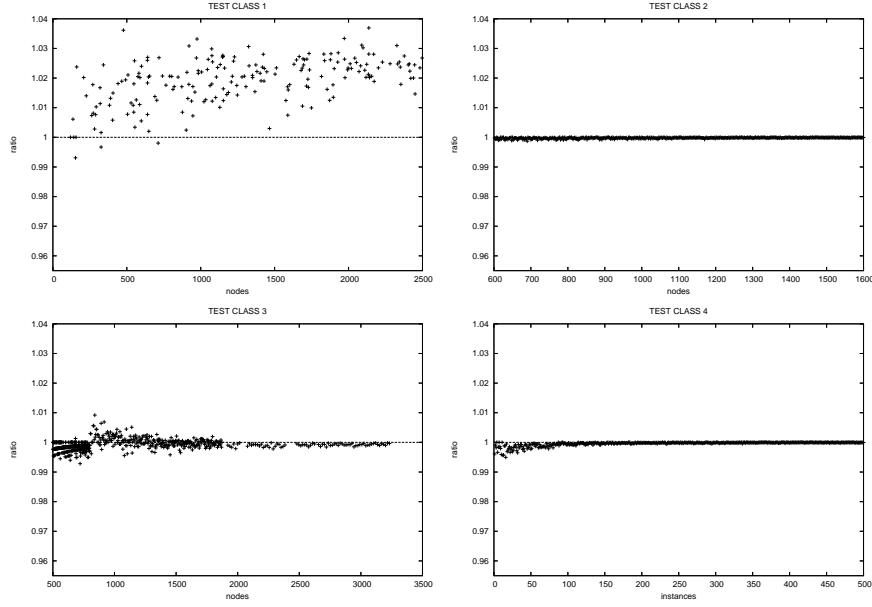


Figure 5.1: Ratio of objective values found by LPcut and by REF to the MAX-DI-CUT instances of the classes 1 to 4.

In an experiment with ID number 3 we only considered the first 642 instances of the third class of instances because the CPLEX solver (of ILOG OPL Studio 3.5) was not able to solve the corresponding linear problems for the further (larger) instances of this class.

To compare the solution values that both techniques found, we plotted the ratios of the cut weights found by our technique and by the reference technique. Let an instance $\{G = (V, A), n = |V|, c_a \in \mathbb{R}_+, \forall a \in A\}$, be the i -th instance of some test class. Assume that for this instance the LPcut and REF found the cuts with weights of C_{LPcut} and $C_{\text{REF}} > 0$, respectively. Let r denote the ratio: $r = \frac{C_{\text{LPcut}}}{C_{\text{REF}}}$. For the instance classes which consist of instances with diverse number of nodes, we plotted the corresponding points (n, r) . But, for the classes that consist of instances with equal number of nodes (or instances with equally sized node sets frequently occur), we plotted the points (i, r) . The Figures 5.1–5.3 show the comparison of two heuristics to the each of test instance classes. All 12 plots have equal range on the y -axis (ratio): $[0.96, 1.04]$.

The experiment with the 4th class of instances indicates that if the graph is denser the ratio of cut values found by two algorithms is closer to 1.0. The instances of this class have uniform weights.

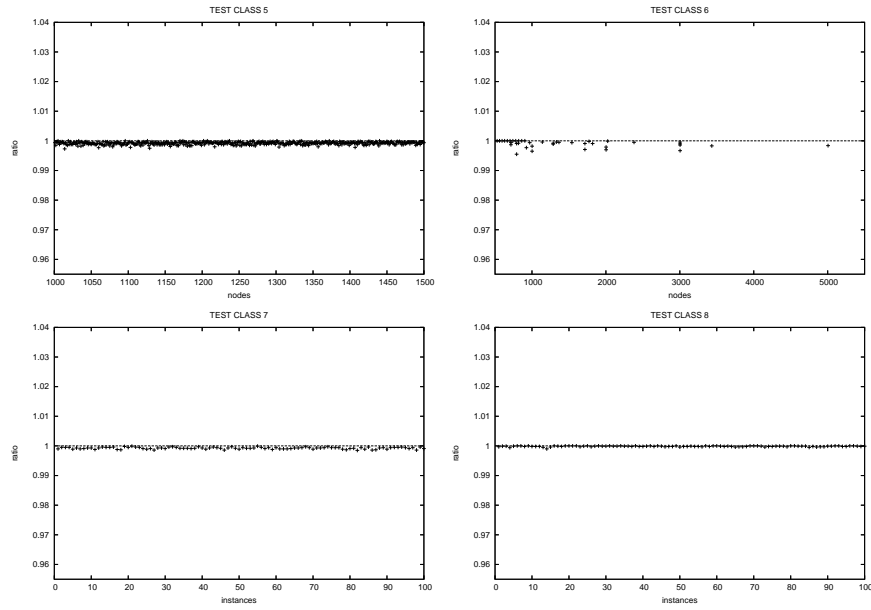


Figure 5.2: Ratio of objective values found by LPcut and by REF to the MAX-DI-CUT instances of the classes 5 to 8.

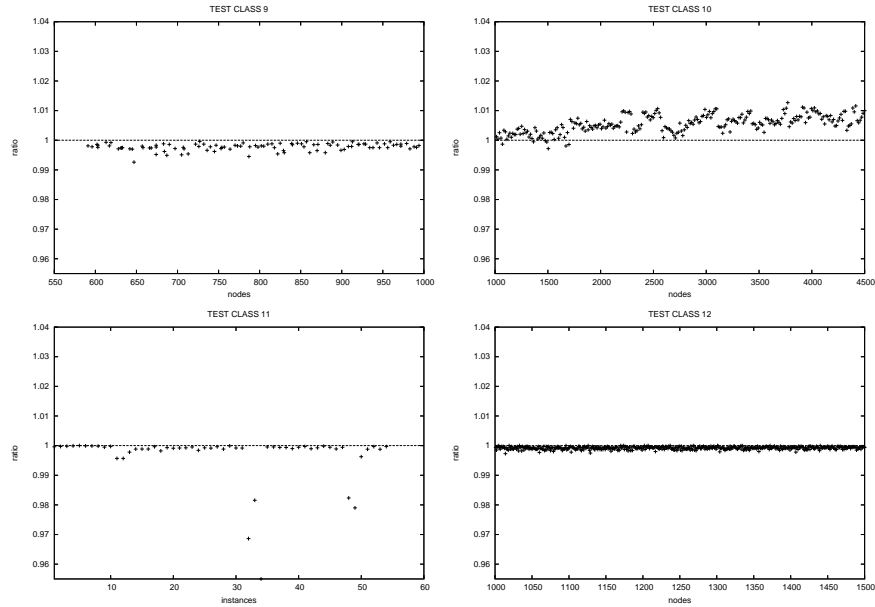


Figure 5.3: Ratio of objective values found by LPcut and by REF to the MAX-DI-CUT instances of the classes 9 to 12.

Comparison of the start and final solutions				
	Ratios of objective values			
ID	the least	90 p.c. \geq	average	the most
(A)	(B)	(C)	(D)	(E)
1	85.35	86.71	87.99	92.28
2	98.59	99.10	99.37	99.73
3	79.81	87.02	90.31	92.98
4	96.63	98.54	99.27	99.85
5	98.33	98.60	98.87	99.45
6	55.95	63.72	83.56	100.00
7	98.00	98.25	98.48	98.91
8	98.70	98.90	99.13	99.51
9	96.02	96.97	97.50	98.60
10	92.96	93.46	94.04	95.48
11	7.33	82.06	91.95	99.47
12	11.21	11.26	11.57	12.15

Table 5.3: Comparison (ratio (p.c.) of values) of the start and final solutions from local search in the LPcut procedure to the MAX-DI-CUT instances. Columns: (A) ID of the experiment; (B) the worst ratio; (C) for 90 p.c. of all instances the ratio hits or exceeds this value; (D) the average ratio; (E) the highest ratio;

Throughout the experiment with all the test instance classes, the worst case solution of our technique occurred in the eleventh class, where the solution value was 95.5 p.c. of value that found by the reference technique. The instances in the eleventh class, of which our technique found the solutions with values at most 98.23 p.c. and at least 95.5 p.c. of solution values found by the reference technique, were all **ruby** type of *toroidal-grid* graphs. Indeed, the results of the twelfth class of instances were confirming the weak performance of our technique in the toroidal-grid type graphs.

Furthermore, we summarize the ratios of our start solutions and the final solutions from local search in Table 5.3. For toroidal grid graph instances of the 11th and all instances of the 12th class, the improvement is dramatic. The ratios of objective values found by LPcut and by REF to these instances were the least among the worst case ratios (Column (B) in Table 5.2), as we mentioned. Another interesting result is that there was no improvement between start and final solutions to 28 p.c. of the instances of the 6th class. To these instances the REF still did not find better solutions than the LPcut.

Evaluation of Iteration to Target Value

For some test instances **LPcut** has found solutions at least as good as **REF**. In the previous experiments the local-search procedure has been run 1,000 times within every run of **REF**. This was indeed quite time consuming. To assess whether 1,000 iterations are actually needed, we ran a couple of tests to answer the reverse question: how many iterations (local-search performances) does **REF** need to find a solution at least as good as solution that **LPcut** found? We run the **TV** procedure by setting the objective values found by the **LPcut** as sub-optimal target values and $q_{max} = 1,000$.

Since the eleventh class of instances contains the graphs of various **rudy** types, we have selected these instances for this evaluation. On each of the instances that were selected for this test, we ran **TV** individually 100 times. The random number generator was initialized in every run with new seed, therefore the runs of procedure were individual in this sense. This produced a distribution of the iteration-to-target-value as shown in Figures 5.4 and 5.5. In these figures the instances were sorted by the sum of corresponding number of iterations to make a smoothly contrasted plot. More formally, two distinct instance numbers x^i and x^j are ordered as $x^i \leq x^j$, $i, j \in \{1, \dots, I\}$ if for the corresponding iteration numbers $\{z_k^i\}$ and $\{z_k^j\}$, $k \in \{1, \dots, N\}$, holds $\sum_{k=1}^N z_k^i \leq \sum_{k=1}^N z_k^j$, where I and N denote the number of instances involved in the test and the number of individual runs of the **TV** procedure, respectively.

In this test we found the following observation:

- for only 2 instances (3.7 p.c. of all test instances), the **TV** did not reach the target values within 1,000 iterations;
- there are another 8 instances (14 p.c. of test instances) for which **TV** required at least 100 iterations to hit or exceed the target values in some runs.
- for 19 instances (35 p.c. of test instances) **TV** required at most 10 iterations to hit or exceed the target values in all runs.

In Figure 5.6, we plotted the distribution of the iteration-to-target-value of all 100 runs of **TV** to all 54 instances. Each of plotted 5,400 points indicate the iterations to target value. For the majority of all runs, the required number of iterations is quite small. However, this is the result of a posteriori inspection. To use **REF** as a proper solver, we had to specify the number of iterations a priori, that is "blindly". Figure 5.6 suggests that approximately 50

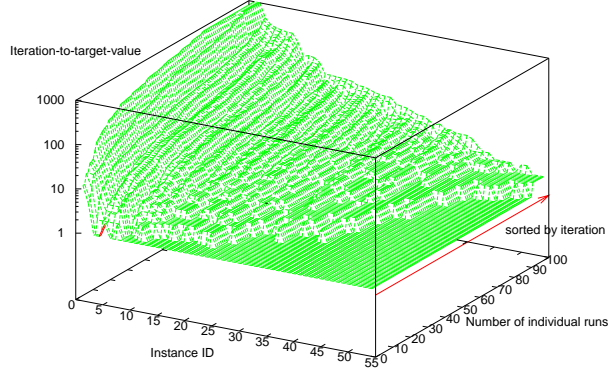


Figure 5.4: The distribution of the iteration-to-target-value in 100 individual runs of TV to 54 MAX-DI-CUT instances of the eleventh class. A point (x, y, z) specifies an instance number x , a number of individual runs y , and a number of iteration-to-target-value z and describes that to the instance x in the y -th run of reference technique the z iterations were needed to find a solution with value at least as good as the target-value.

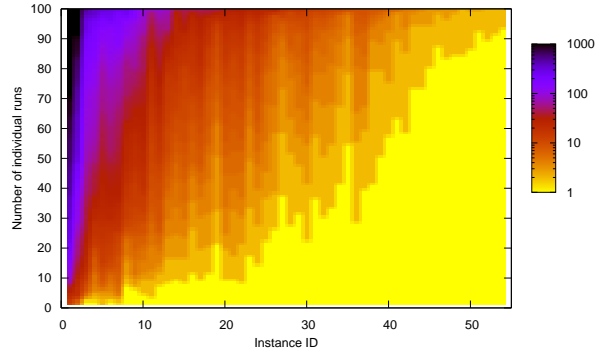


Figure 5.5: The map of the distribution of the iteration-to-target-value shown in Figure 5.4. A color on a point (x, y) indicates the number of iterations that were needed to find a solution with a value at least as good as the target-value to the instance x in the y -th run of TV.

iterations would be necessary to match LPcut in 90 p.c. of all runs. The distributions of the iteration-to-target-value for each of the 54 instances are shown in the Figures B.1–B.5, which can be found in Appendix B.1.

Furthermore, we investigated the distribution of iteration-to-target-value in the 100 individual runs of TV to the 32 instances of the first class. Note that the first class contains

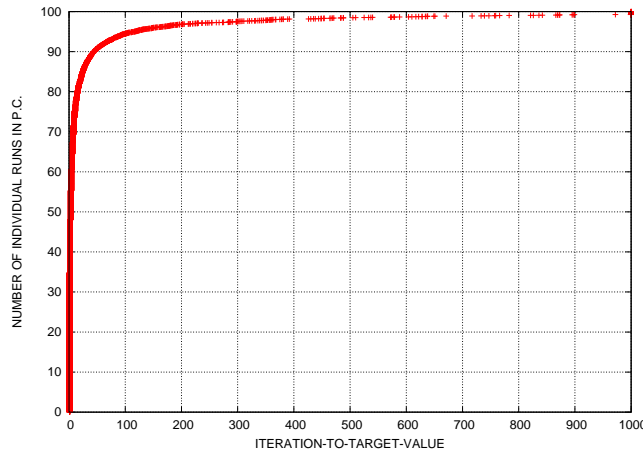


Figure 5.6: The distribution of the iteration-to-target-value by 100 runs of **REF** to all 54 instances of the eleventh class of **MAX-DI-CUT**. Hence, there are 5,400 points in the plot. The number of iterations are sorted and associated with the percentage of total data to display.

randomly weighted graphs. The experiment produced the distribution of the iteration-to-target-value as shown in Figure 5.7. In the plot, the instances are sorted by the sum of the corresponding number of iterations, similarly to the previous experiment, to make a smoothly contrasted picture.

In this experiment we found the following observation:

- for 22 instances (70 p.c. of all instances), all runs of **TV** could not reach the target values within 1,000 iterations;
- for another 6 instances, there were some **TV** runs which could not reach the target values within 1,000 iterations;
- there are only 3 instances for which **TV** required at most 200 iterations to hit or exceed the target values in all runs.

In Figure 5.8, we plotted the distribution of the iteration-to-target-value in all 100 runs of **TV** to all 32 instances. In contrast to the previous experiment (with the twelfth class of instances), the numbers of iterations to target value were less than 1,000 in less than 20 p.c. of runs in this test. Figure 5.8 suggests that 1,000 iterations would not be enough for the reference strategy to match our technique in about 80 p.c. of all runs. For the majority of all runs **TV** could not find target solutions in 1,000 iterations.

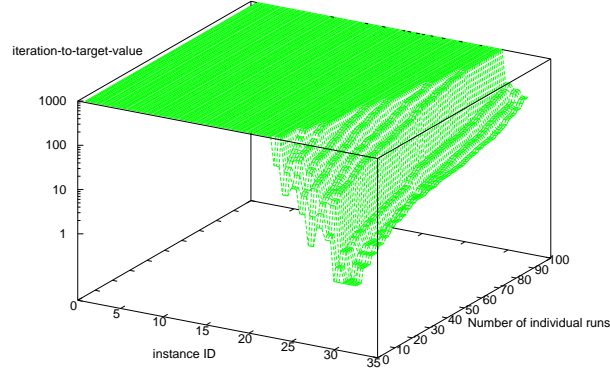


Figure 5.7: The distributions of the iteration-to-target-value by 100 runs of TV to 32 MAX-DI-CUT instances of the first class. A point (x, y, z) specifies an instance number x , a number of individual runs y , and a number of iteration-to-target-value z and describes that for the instance x in the y -th run of TV z iterations were needed to find a solution with value at least as good as the target-value.

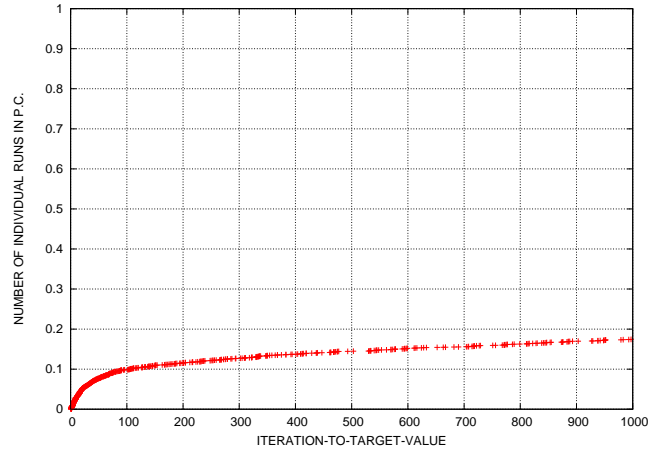


Figure 5.8: The distribution of the iteration-to-target-value by 100 runs of REF to all 32 instances of the first class of MAX-DI-CUT. The number of iterations are sorted and associated with the percentage of total data to display.

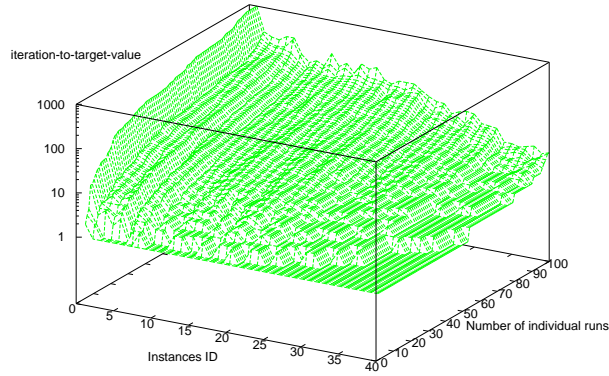


Figure 5.9: The distribution of the iteration-to-target-value by the 100 individual runs of TV to all instances of the sixth class of MAX-DI-CUT. A point (x, y, z) specifies an instance number x , a number of individual runs y , and a number of iteration-to-target-value z and describes that in the y -th run of TV to instance x , the z iterations were needed to find a solution with value at least as good as the target-value.

Furthermore, we investigated the iteration-to-target-value in the further 100 individual runs of TV to 40 instances of the sixth class. Note that the sixth class contains uniformly weighted graphs of *rud*-type simplex. The produced distribution of the iteration-to-target-value is shown in Figures 5.9 and 5.10. In this test we found the following observation:

- for only one instance, there were 7 runs of TV which could not reach the target value within 1,000 iterations;
- there are other 12 instances (30 p.c. of test instances) for which TV required at least 100 iterations to hit or exceed the target values in some runs.
- for 7 instances (17.5 p.c. of test instances) TV required at most 10 iterations to hit or exceed the target values in all runs.

Conclusion: Our technique and the reference technique have found cut values that are very close to each other continuously in all test instance classes, even though two algorithms differ sufficiently from each other. The only plausible explanation for this empirical outcome is that two algorithms have found near-optimal solutions.

The comparison of start and final solutions of local search in the LPcut procedure shows

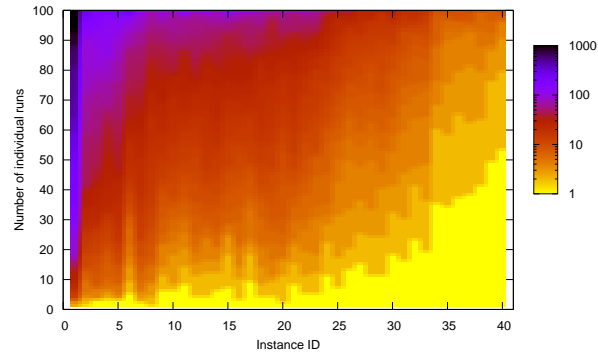


Figure 5.10: The map of the distributions of the iteration-to-target-value shown in Figure 5.9. A color on a point (x, y) indicates the number of iterations that were needed to find a solution with a value at least as good as the target-value for instance x in the y -th run of TV.

that the start solution delivered by the LP and the local search starting from this solution – the hybridization – result our strong empirical findings.

5.3. Maximum Satisfiability

This section devotes to the empirical study of our technique for the three variations of the MAX- k -SAT problem with $k \in \{2, 3, 4\}$. In each case, we describe the generation of various test instances for the experiments and then present the computational results of experiments carried out.

We describe the random instance generators and some issues in the generation of test instances as follows.

5.3.1. Generation of Instances

We consider the instances of the MAX-2-SAT problem in the graphical representation. For a given graph, the edge represents the clause with two literals, and the literals are the nodes connected by this edge. The set of edges is the set of clauses, and the set of variables is the set of boolean variables and their negations. For the nonlinear formulation of the MAX- k -SAT, $k \geq 2$, problem there were additional variables introduced, which represent the negations of the boolean variables (Section 4.3). Therefore, the number of nodes of the graph that represents the problem statement is twice the number of boolean variables. The graph generator `rud` as well as our self written (the second) graph generator, which were used to generate the instance classes of the MAX-DI-CUT problem, were utilized to generate the instances of MAX-2-SAT problem. In addition, we used a random graph generator written by Jagota and Sanchis [95] that generates graphs with known sizes of cliques (actually designed for the max-clique problem).

To generate an instance of the MAX-3-SAT problem, first we generated the instances of the MAX-2-SAT, then we extended the clauses into the length of 3 (literals). We proceed as follows. Assume that we are going to construct the instance of MAX-3-SAT with a set of variables X and a set of clauses Z . We are given a graph $G = (V, E)$, $n = |V|$, $m = |E|$, and n be *even*. Also, for each edge $e \in E$ a nonnegative weight $c_e \in \mathbb{R}^+$ is given. The set of nodes $V = \{v_1, \dots, v_n\}$ is a set of literals. The literals are the boolean variables and their negations. Then $X = \{v_1, \dots, v_{\frac{n}{2}}\}$, and an edge $e = (v, w) \in E$ is the clause with two literals v and w , $Z = E$. We denote a set of negations of boolean variables $\overline{X} = \{\overline{v_1}, \dots, \overline{v_{\frac{n}{2}}}\}$. First, we set the number of clauses with three literals at random $r \in [\lfloor \frac{1}{2}m \rfloor, m]$. But at least half of the clauses are extended to length of three. Then iteratively, we select the clause $C_i \in Z$ with length 2 at random and append (assign) random literal $x_j \in X \cup \overline{X}$, $x_j \notin C_i$, which

makes C_i to length of three: $C_i = C_i \cup \{x_j\}$. We repeat (or iterate) until a desired number, r , of clauses of length three are generated. At last we eliminate the duplications of identical clauses.

We used some benchmark class of the MAX-3-SAT instances¹ in our experiments. Additionally, in the experiments for the MAX-3-SAT and MAX-4-SAT we utilized a CNF-formula generator of B. Selman [142], which generates random instances of MAX- k -SAT problem, $k \geq 2$. We call this generator Selman's generator.

Moreover, the random graph generator **GTgraph** [11] is used to generate some classes of the MAX-4-SAT instances.

5.3.2. Max-2-SAT

In this section we give our attention to the empirical study of our technique for the MAX-2-SAT problem.

Test Instances

We considered seven classes of test instances. Table 5.3.2 shows the details of the instances of each class (in graphical representation): a number of instances in that class, a number of nodes (or range), a density (or range), a construction of edge weights, and which generator has been used.

The instance classes with ID numbers 1, 2 and 3 consist of uniformly weighted graphs. The first class contains 50 graphs with size of 100 to 3,000 nodes. The instances of the second and the third classes are the graphs of **rudy**-type simple-random. The graphs of each of the second and third classes vary in number of nodes, but equal in density, where the instances of the second class have 5 p.c. density, and the third class have (sparser) instances with 1 p.c. density. The further classes of instances consist of diversely weighted graphs. The instances of the 4th class have weights that are independently and randomly chosen from a uniform distribution of a range $[1, \dots, 100]$. The instances of the 5th class have more varied weights that are independently and randomly chosen from a uniform distribution of a range $[1, \dots, 10000]$. Similarly, the weights of each instance of the 6th class have been chosen randomly and independently from a range $[1, \dots, 3000]$. The last class is exactly same as the 11th class in Section 5.2.1, which was generated for the MAX-DI-CUT problem,

¹[ftp://dimacs.rutgers.edu/pub/challenge/satisfiability/contributed/iwama](http://dimacs.rutgers.edu/pub/challenge/satisfiability/contributed/iwama)

class ID	number of instances	number of nodes	density	arc weight	generator type
1	50	[100–3000]	[0.30–70.00]	uniform	<i>clique</i> ^a
2	54	[800–3000]	[0.12–6.00]	various	rud y:various types ^b
3	500	[600–1600]	5.00	uniform	rud y:simple-random
4	500	[1500–2500]	1.00	uniform	rud y:simple-random
5	100	2000	0.59	uniform distr. of [1, . . . , 100]	rud y:almost-planar
6	100	800	6.00	uniform distr. of [1, . . . , 10000]	rud y:simple-random
7	300	[1200–5400]	[0.11–0.52]	uniform distr. of [1, . . . , 3000]	self-written ^c

^agenerated by a public domain graph generator written by Jagota and Sanchis [95], which is available from a website of the second DIMACS implementation challenge: [fpt://dimacs.rutgers.edu/pub/challenge/](http://fpt.dimacs.rutgers.edu/pub/challenge/)

^bsimilar to G-set (see Section 5.2.1).

^c“nearly planar” graphs generated by self written generator (see Section 5.2.1).

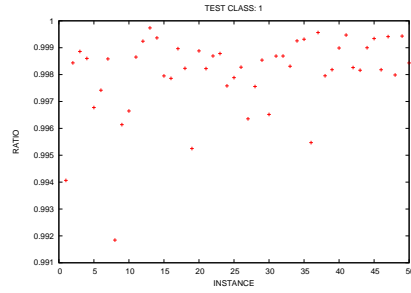
Table 5.4: The various classes of test instances for the MAX-2-SAT problem

with an exception that we omitted the orientation of the edges.

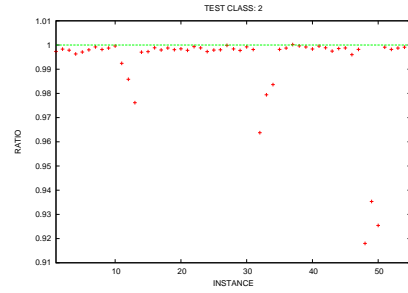
Experimental Evaluation

The first experiment consists of seven parts, each part considering one class of the test instances. To compare the results of two heuristics we plotted the ratios of objective values found by our technique and by the reference technique in Figures 5.11a–5.11g. In none of the instances of all test classes, were the ratios worse than 3.0 p.c., except in the toroidal grid instances of the second class. We can see from the Figure 5.11 that the distribution of the ratios is stable in the classes of uniformly weighted instances, as well as, in the classes consisting of instances with diverse weights.

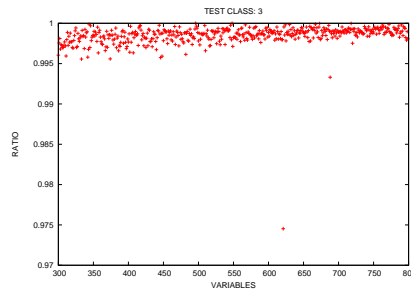
We summarize the experimental results that are the comparison of objective values and the comparison of CPU times in Table 5.5. The descriptions of columns of this table are analogous to those of the Table 5.2. The worst case solutions (Column (B)) of our technique have been occurred during the test with the second class of instances, specifically, to the instances of **rud**y-type toroidal-grid. In this class, **LPcut** found the solution with value at most 8.2 p.c. less than the **REF** found. But, for a majority (90 p.c. of instances) of the



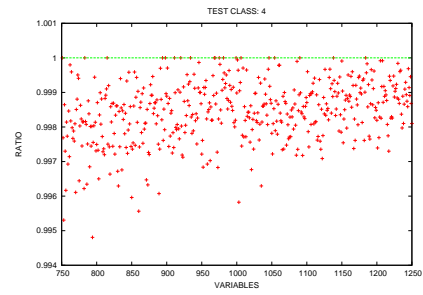
(a) Instance Class 1



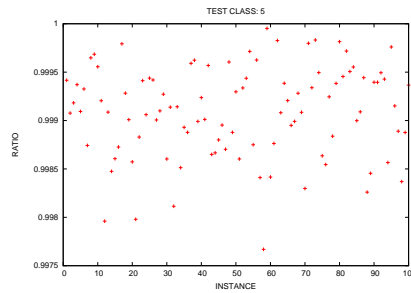
(b) Instance Class 2



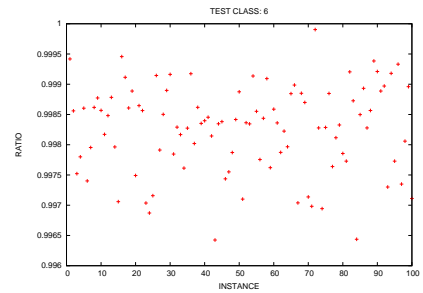
(c) Instance Class 3



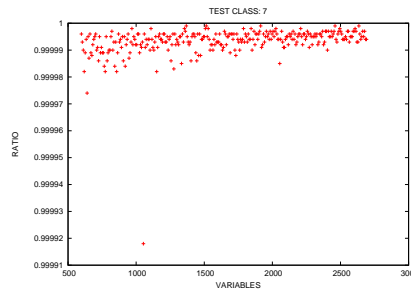
(d) Instance Class 4



(e) Instance Class 5



(f) Instance Class 6



(g) Instance Class 7

Figure 5.11: The ratio of objective values found by LPcut and by REF to the MAX-2-SAT instances.

		Comparison of objective values				Comparison of CPU times		
ID	the least	90% \geq	average	the most	percentage	the most	90% \leq	average
(A)	(B)	(C)	(D)	(E)	(F)	(G)	(H)	(I)
1	99.18	99.63	99.80	99.97	0.0	0.2260	0.1198	0.0480
2	91.80	97.60	99.08	100.018	2.27	0.0045	0.0031	0.0018
3	97.45	99.73	99.85	100.0042	0.4	0.0170	0.0107	0.0400
4	99.48	99.74	99.85	100.15	3.6	0.0208	0.0057	0.0047
5	99.76	99.84	99.91	99.99	0.0	0.0029	0.0024	0.0022
6	99.64	99.71	99.82	99.99	0.0	0.0105	0.0102	0.0095
7	99.99	99.9989	99.9993	99.9999	0.0	0.0045	0.0023	0.00126

Table 5.5: Comparison of our technique with the reference technique for MAX-2-SAT problem. Columns: (A) ID of the experiment; (B)–(E) ratios (p.c.) of the objective values found by **LPcut** and by **REF**: (B) the worst, (C) for 90 p.c. of all instances the ratio (p.c.) hits or exceeds this value, (D) the average, (E) the best; (F) the percentage of instances where the outcome of **LPcut** was better than or equal to the outcome of the **REF**; (G)–(I) ratios of the CPU times needed by **LPcut** and the **REF**: (G) the most; (H) for 90 p.c. of all instances the ratio matches or does not exceed this value; (I) the average.

instances the difference is less than 3.0 p.c. The details of this part of experiment can be found in a Table B.1 of Appendix B.2.

Furthermore, in Table 5.6 we summarize the comparison of the start and final solutions of local search from the **LPcut** procedure. The worst case ratio (Column (B)) was no more than 18 p.c., and the 90 p.c. quantile (Column (C)) was smaller than 16 p.c. Except for the second instance class, the 90 p.c. quantile was 3.2 p.c. or smaller. In other words, in the 90 p.c. of instances the improvement through local search was at most 3.2 p.c. On the other hand, in none of the instance classes was the 90 p.c. quantile of the ratios of objective values found by the **LPcut** and by the **REF** more than 0.37 p.c., except the second instance class (Column (C)) of Table 5.5). The least ratios of the start and final solutions of the **LPcut** have occurred to the toroidal grid instances of the second class, where the **LPcut** found worst solutions compared to the **REF**, respectively.

Evaluation of Iteration To Target Value

We did further experiments to test the quality (of run time) of our technique. Since our technique could not find a solution at least as good as the reference technique has found

Comparison of the start and final solutions				
	Ratio of objective values			
ID	the least	90 p.c. \geq	average	the most
(A)	(B)	(C)	(D)	(E)
1	97.99	98.48	99.16	99.80
2	82.57	84.38	95.74	98.93
3	94.70	97.82	98.23	98.88
4	96.52	96.87	97.21	97.83
5	97.57	97.82	98.04	98.40
6	97.34	97.51	97.83	98.37
7	99.34	99.48	99.60	99.83

Table 5.6: Comparison (ratio (p.c.) of values) of the start and final solutions from local search in the LPcut procedure to the MAX-2-SAT instances. Columns: (A) ID of the experiment; (B) the worst ratio (p.c.); (C) for 90 p.c. of all instances the ratio (p.c.) hits or exceeds this value; (D) the average ratio (p.c.); (E) the highest ratio (p.c.);

to the instances of some classes, we did an experiment, an investigation of the iteration-to-target-value, analogously to the experiments that we did for the MAX-DI-CUT problem (Section 5.2.2). In these experiments we ran the TV procedure with a setting: the target values are the solution values found by our technique and $q_{max} = 1,000$. Remember, that the TV procedure is the same as the REF procedure which stops when the target value is reached.

For the first evaluation we have selected 6 instances of the 2nd class of instances with ID-numbers of 3, 11, 16, 22 and 23. To these instances our technique has found the objective values of 15360, 787, 4381, 16672 and 16654, respectively.

We ran the TV procedure with the above mentioned setting individually 200 times to each of instances with ID numbers 3, 11 and 16. The produced distributions of the iteration-to-target-value are shown in Figure 5.12. In Figure 5.12 an i -th sorted iteration number (k_i) is associated with a number of individual run i , and the points (k_i, i) are plotted, for all $i = 1, \dots, 200$. TV needed less than 80 iterations to match our technique to these instances.

To the instances with ID numbers 22 and 23 we ran the TV only 100 times, owing to an extensive computing time. We plotted in Figure 5.13 the produced distributions of the iteration-to-target-value. Figure 5.13 suggests that the TV needed more than 100 iterations to find the solutions with values more than or equal to the values of our technique's solutions in at least 40 p.c. of runs. Even to the instance with ID number of 22, TV could not reach

solution value found by our technique in 1,000 iterations in 4 p.c. of runs.

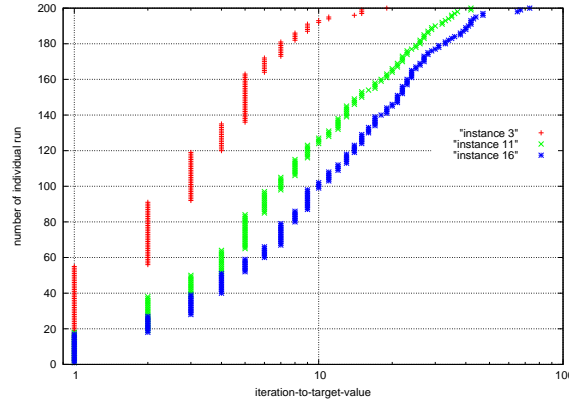


Figure 5.12: The distributions of the iteration-to-target-value in 200 individual runs of TV to the instances 3, 11 and 16 of the second class of MAX-2-SAT.

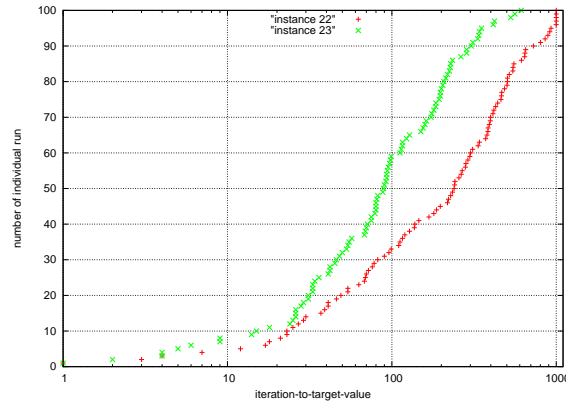


Figure 5.13: The distributions of the iteration-to-target-value in 100 individual runs of TV to the instances 22 and 23 of the second class of MAX-2-SAT.

We evaluated the distribution of iteration-to-target-value in further classes of instances. First, we selected the smallest 50 instances of the 4th class of instances (we limited the number of instances to 50, because of extensive computing time). To each of these instances, we ran TV individually 60 times. The produced distributions of the iteration-to-target-value are shown in Figures 5.14 and 5.15. In Figures 5.14 and 5.15, the instances are sorted by the corresponding average number of iterations to make a smoothly contrasted plot.

In this test we found following observation:

- on 14 p.c. of test instances, there were some runs of TV which could not reach the

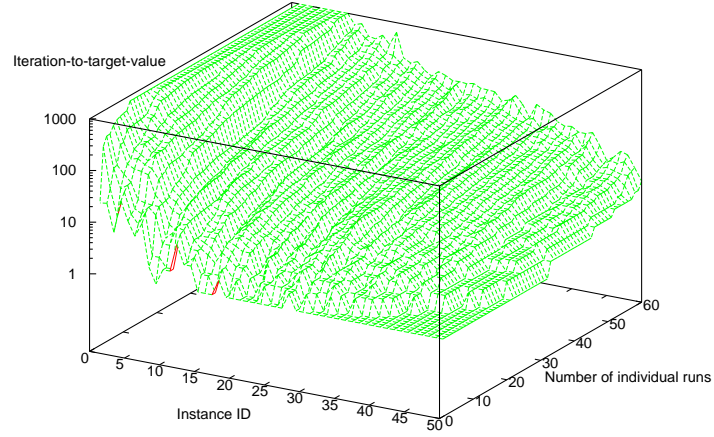


Figure 5.14: The distribution of the iteration-to-target-value in 60 individual runs of TV to 50 MAX-2-SAT instances of the 3rd class. The point (x, y, z) specifies the instance number x , the number of individual runs y , and the number of iteration z , and describes that y -th run of TV to instance x , the z iterations were needed to find a solution with value at least as good as the target-value.

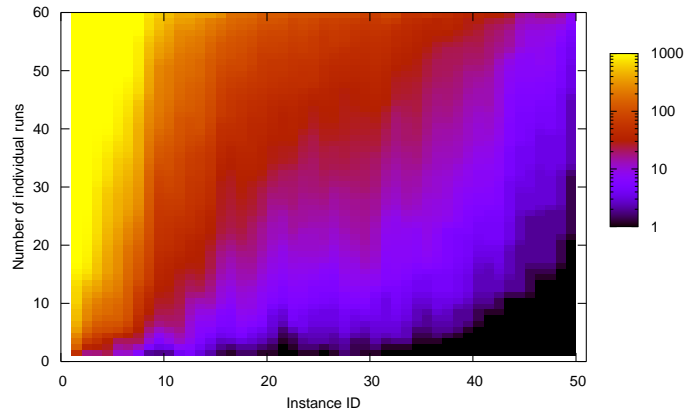


Figure 5.15: The map of the distributions of the iteration-to-target-value, which are shown in Figure 5.14. A color on a point (x, y) indicates the number of iterations that was needed to find a solution with value at least as good as the target-value to instance x in the y -th run of the TV.

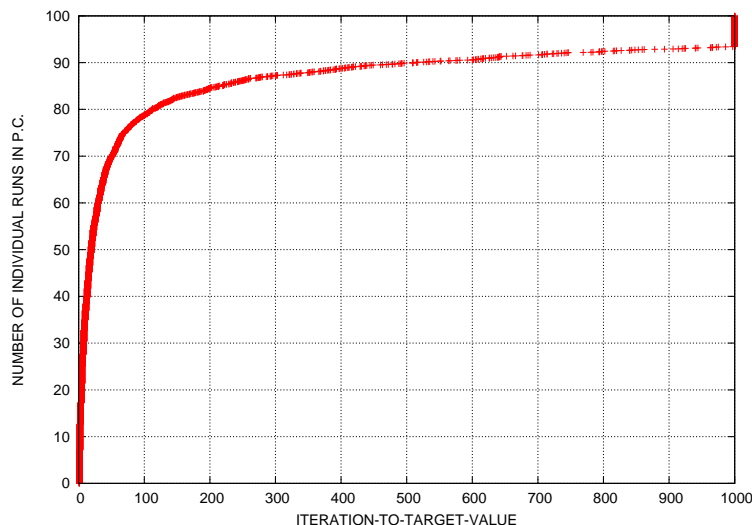


Figure 5.16: The distribution of the iteration-to-target-value in 60 runs of the TV to all 50 MAX-2-SAT instances of the 3rd class. The number of iterations are sorted and associated with the percentage of total data to display (3,000 runs).

target values within 1,000 iterations;

- to 12 instances (24 p.c. of test instances) the TV required at least 100 iterations to find objective values better than or equal to the target values in at least 33 p.c. of the runs.

In Figure 5.16 we plotted the distribution of the iteration-to-target-value in 60 runs of the TV to all 50 instances. For the majority of all runs, the required numbers of iterations is not small. Figure 5.16 suggests that approximately 500 iterations would be necessary for the REF to match LPcut in 90 p.c. of all runs.

The distributions of the iteration-to-target-value to each of the 50 instances are shown in detail in Figures B.6–B.9, which can be found in Appendix B.3.

In this (recent) test there were instances to which TV could not find solutions that were at least as good as LPcuts'. Therefore, we were interested to test whether this behavior of the reference technique was influenced by the solutions of our technique: how many iterations does the TV need to find a solution with value at least as good as the *random local optimum*? In other words, would it come to the same result if we choose the arbitrary suboptimal target value? As a *random local optimum* we call the solution found by running the REF procedure with only one iteration (i.e., one run of local search starting from random solution). We

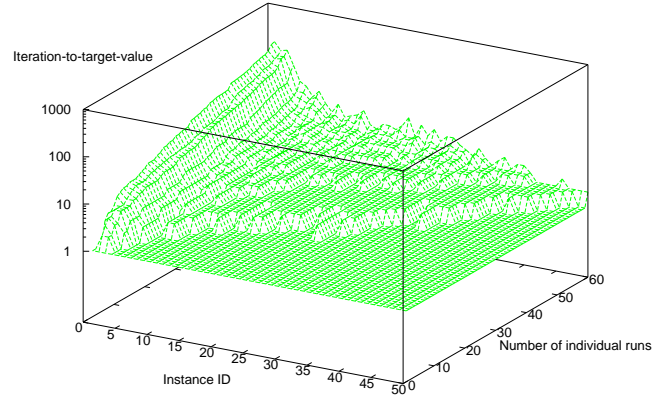


Figure 5.17: The distributions of the iteration-to-target-value by 60 individual runs of TV to 50 MAX-2-SAT instances of the third class. A point (x, y, z) specifies an instance number x , a number of individual runs y , and a number of iteration-to-target-value z , and describes that to instance x in the y -th run of TV the z iterations were needed to find a solution with value at least as good as the target-value.

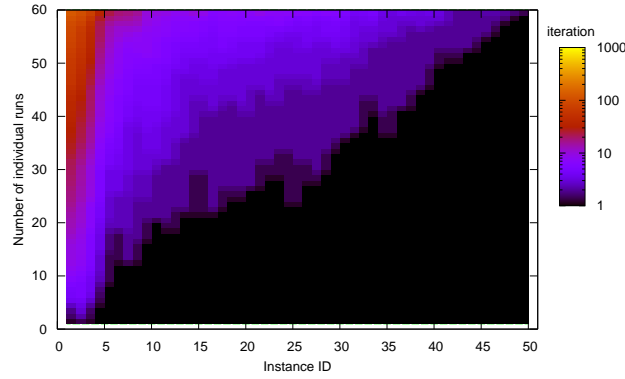


Figure 5.18: The map of the distributions of the iteration-to-target-value, which are shown in Figure 5.17. A color on point (x, y) , an instance number x and a number of individual run y , indicates the number of iterations that were needed to find a solution with value at least as good as the target-value to instance x in the y -th run of TV.

found the random local optimum to each of the previously selected 50 instances, then ran the TV procedure individually 60 times, by setting these random local optima as target

values, to each of selected instances. This generates another distribution of the iteration-to-target-value. Here, the difference to the previous test is only the chosen target-values. The Figures 5.17 and 5.18 summarize the generated distributions corresponding to 50 instances in 60 individual runs of TV.

In contrast to the previous test, the all runs of TV have found the solutions with values better than or equal to the target values within 180 iterations to all instances. In fact, to only 3 instances some TV runs required more than 100 iterations to beat the target values; to other 7 instances some TV runs required more than 10 iterations to beat the target values. To the rest of instances, which is about 80 p.c. of all instances, no run of TV required more than 10 iterations to beat the target values.

By comparing these results with results of the previous experiment, we can see that our technique finds better solutions than random local optima. The reference technique and our technique produce the solutions whose objective values do not differ from each other more than a small percentage, even though two heuristics differ sufficiently from each other. Consequently, we draw a conclusion that our technique finds near-optimal solutions to the MAX-2-SAT problem.

Comparison to Theoretical Bounds

For a certain type of MAX-2-SAT instances theoretical bounds of the expected optimal objective value are available. Recall Theorem 3.3.1 which states the asymptotical bounds of the expected optimal objective value of a certain type of instance [35]. We compared our empirical results with these bounds.

Each of the third and the fourth test classes consists of 500 random instances that are generated analogously to the the random instance model that is stated in Theorem 3.3.1. For the 3rd class we have $n \in [300, \dots, 800]$ and $30 \leq c \leq 80$, and for the 4th class we have $n \in [750, \dots, 1250]$ and $15 \leq c \leq 25$.

We compared the empirical values, which are the solution values found by the LPcut to these instances, to the two approximations, which are the asymptotic lower and upper bounds of the expected optimal value, respectively. The upper bound values are important for our point. The lower bound is displayed only for orientation. We omitted the indefinite fraction $o_c(1)$ in the lower bound calculation.

We plotted the ratios (p.c.) of objective value found by the LPcut and theoretical lower and upper bounds for each of instances in the 3rd and 4th classes in Figure 5.19.

Comparison of empirical values by LPcut and theoretical bounds to the MAX-2-SAT instances

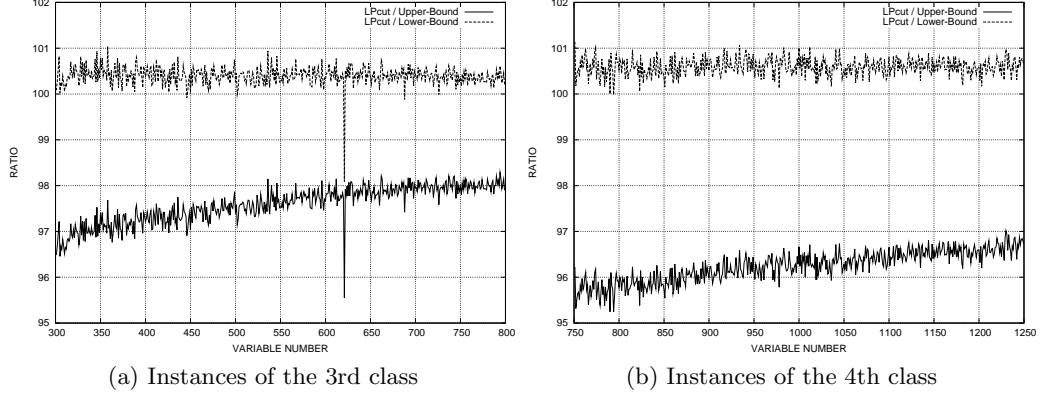


Figure 5.19: The ratio (p.c.) of objective value found by the LPcut and theoretical lower bound, and the ratio of objective value found by the LPcut and theoretical upper bound

Comparison of empirical values found by REF and theoretical bounds to the MAX-2-SAT instances

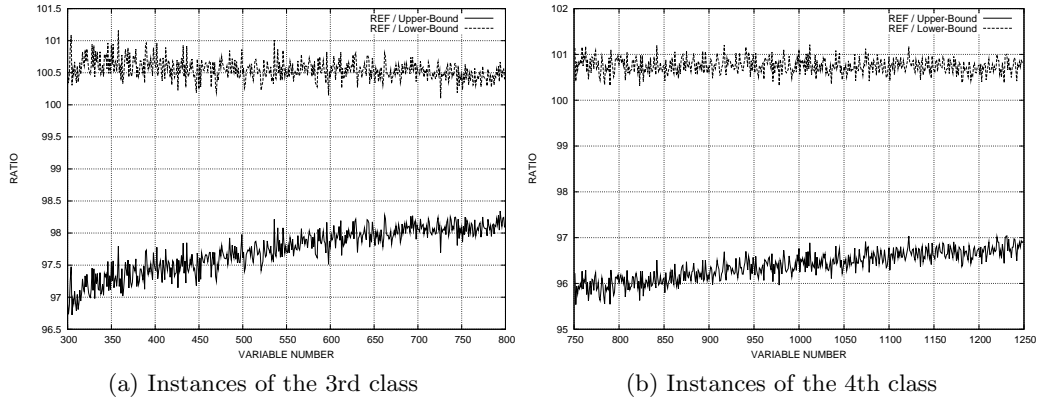


Figure 5.20: The ratio (p.c.) of objective value found by the REF and theoretical lower bound, and the ratio of objective value found by the REF and theoretical upper bound

Furthermore, we compared the empirical values found by the REF to these instances to the two approximations, which are the asymptotic lower and upper bounds of the expected optimal value, respectively. The comparison, the ratio (p.c.) of objective value found by the REF and theoretical lower and upper bounds, is plotted in Figure 5.20.

The objective values found by LPcut to the instances of 3rd class differ from the theoretical upper bound by at most 3.5 p.c., and from the theoretical lower bound by less than 1.0 p.c. (with one exception in each case). For the 4th class the ratio of the solution value

found by the LPcut and the theoretical upper bound is at most 4.8 p.c.

This outcome requires an explanation. Since the lower and upper bounds have no methodical relations with our algorithm, in our opinion, the only plausible explanation is that these bounds are empirically valid actually for even small values of c , and our algorithm have found near optimal solutions.

5.3.3. Max-3-SAT

This section is devoted to the experimental results of our technique for the MAX-3-SAT problem. We first describe the test instances, then the empirical results.

Test Instances

class ID	number of instances	number or range of nodes	number or range of clauses	clause weight
1	500	[250–750]	[2500–10000]	uniform
2	500	[500–1000]	[1250–5000]	various
3	500	[250–750]	[3224–28827]	uniform
4	500	[500–1000]	[2500–10000]	uniform distr. of $[1, \dots, 100000]$
5	300	[500–2600]	[3158–15855]	uniform distr. of $[1, \dots, 2200]$
6	500	500	[1424–38811]	uniform
7	100	650	7630	uniform distr. of $[1, \dots, 100]$
8	100	650	7630	uniform distr. of $[1, \dots, 100000]$
9	54	[400–1500]	[800–20200]	various
10	48	[50–200]	[80–1200]	uniform ^a
11	23	[50–200]	[80–400]	uniform ^b
12	500	[500–1000]	[2140–4266]	uniform

^asatisfiable instances

^bnot satisfiable instances

Table 5.7: Various classes of test instances for the MAX-3-SAT problem

We considered twelve classes of test instances. Table 5.7 shows the characteristics of each class. The instances of the classes from 1 to 9 have been generated through extending the MAX-2-SAT instances as we mentioned in Section 5.3.1. The graphs representing the instances of MAX-2-SAT have been generated by the `rud` as well as by self written generator (independently of the instances that were generated in the experiments for MAX-2-SAT).

The instances of the classes 10 and 11 are satisfiable and not satisfiable instances of

ID	Comparison of objective values				Comparison of CPU times		
	the least	90% \geq	average	the most	the most	90% \leq	average
(A)	(B)	(C)	(D)	(E)	(F)	(G)	(H)
1	98.11	98.67	99.05	99.76	0.0059	0.0052	0.0046
2	98.12	98.54	98.97	100.00	0.0498	0.0073	0.0056
3	98.13	98.98	99.32	99.95	0.0055	0.0046	0.0040
4	97.92	98.68	99.04	99.75	0.0058	0.0047	0.0040
5	97.16	97.80	98.12	99.09	0.2975	0.2538	0.1204
6	95.10	97.05	98.06	99.23	0.0987	0.0779	0.0701
7	98.52	98.67	98.92	99.39	0.0053	0.0046	0.0044
8	98.46	98.65	98.88	99.33	0.0048	0.0046	0.0044
9	96.19	97.66	98.82	99.79	0.1836	0.1330	0.0743
10	94.67	96.34	97.70	100.00	0.0886	0.0396	0.0227
11	95.00	96.25	97.97	99.37	0.0381	0.0337	0.0233
12	97.90	98.66	99.09	100.03	0.0122	0.0058	0.0046

Table 5.8: Comparison of our technique with the reference technique for the MAX-3-SAT problem. Columns:

(A) ID of the experiment;

(B)–(E) ratios (p.c.) of the objective values found by LPcut and by REF: (B) the worst, (C) for 90 p.c. of all instances the ratio (%) hits or exceeds this value, (D) the average, (E) the best;

(F)–(H) ratios of the CPU times needed by LPcut and the REF: (F) the most; (G) for 90 p.c. of all instances the ratio does not exceed this value; (H) the average.

Asahiro et al. [9]¹, respectively. The instances of the twelfth class have been generated by the Selman’s generator.

Experimental Evaluation

The experiment consists of twelve parts, each part is considering one class of the test instances. We summarize the computational results, the comparison of objective values and comparison of CPU times, in a Table 5.8.

In none of the instance classes was the worst case ratio (column (B)) worse than 5.4 p.c. The 90 p.c. quantile (column (C)) were nearly 3.75 p.c. in instance classes 10 and 11, and were smaller than 3 p.c. in all other instance classes.

Table 5.9 summarizes the ratio of start solution value and final solution value of the local search in the LPcut procedure. In all instance classes the worst case ratio (Column (B) of

¹[ftp://dimacs.rutgers.edu/pub/challenge/satisfiability/contributed/iwama](http://dimacs.rutgers.edu/pub/challenge/satisfiability/contributed/iwama)

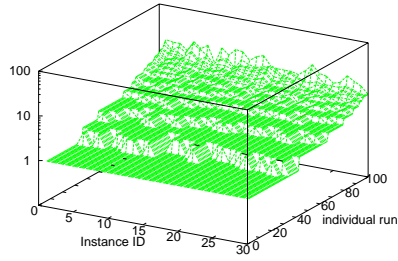
Discrepancy of the start and final solutions				
	Ratios of objective values			
ID	the least	90% \geq	average	the most
(A)	(B)	(C)	(D)	(E)
1	96.38	97.15	97.56	98.35
2	96.17	96.79	97.24	98.38
3	97.35	98.08	98.48	99.13
4	96.36	96.95	97.33	97.98
5	95.08	95.80	96.64	97.99
6	94.97	97.46	98.17	99.03
7	98.95	99.10	99.26	99.56
8	98.86	99.06	99.24	99.54
9	96.91	97.66	98.70	99.55
10	93.91	95.35	96.96	98.95
11	93.38	94.79	96.43	98.45
12	95.93	96.72	97.15	98.01

Table 5.9: Comparison (ratio (p.c.) of values) of the start and final solutions from local search in the LPcut procedure to the MAX-3-SAT instances. Columns: (A) ID of the experiment; (B) the worst; (C) for 90 p.c. of all instances the ratio hits or exceeds this value; (D) the average; (E) the highest;

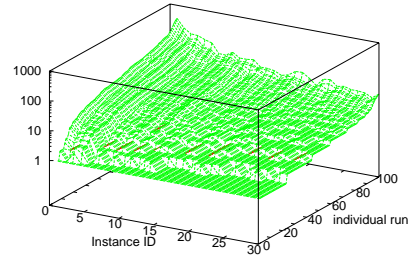
Table 5.9) was nearly 7 p.c. or smaller; the 90 p.c. quantile (Column (C) of Table 5.9) was nearly 5.2 p.c. or smaller. In other words, local search was able to improve the start solution at most nearly by 5.2 p.c. in 90 p.c. of instances. Even the local search slightly improved the start solutions, the differences of solution values found by the LPcut and by the REF were marginal.

Evaluation of Iteration to Target Value

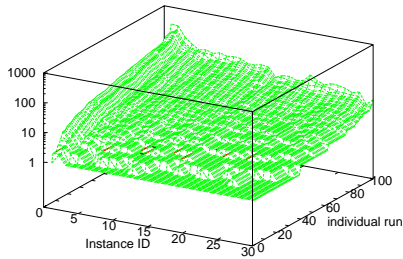
To test the qualities of (run-time of) our approach and the test instances, we did further experiment, an investigation of the iteration-to-target-value, analogously to the experiment we did for the MAX-2-SAT problem (Section 5.3.2). To do the experiment in a moderate computing time we selected only 30 instances with the smallest number of variables of each class, except the 11th class, which contains only 23 instances. To each of these instances we ran REF independently 100 times by setting the solution values found by the LPcut as the target values. This produces the distributions of the iteration-to-target-value as shown in the Figures 5.21 and 5.22. In the plots the instances are sorted by the sum of



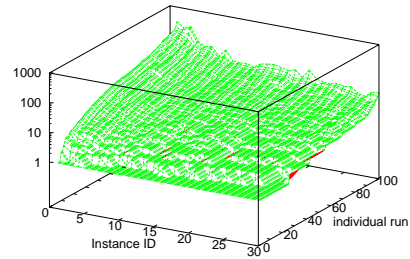
(a) Instance Class 1



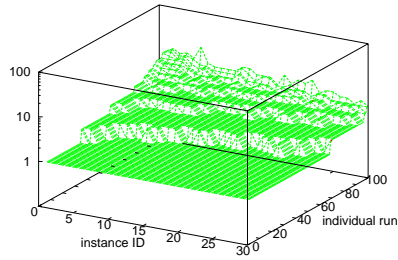
(b) Instance Class 2



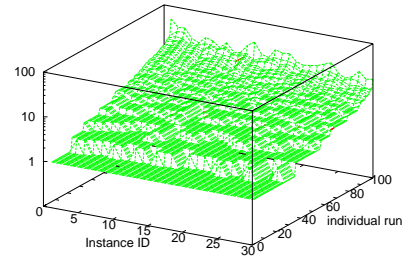
(c) Instance Class 3



(d) Instance Class 4



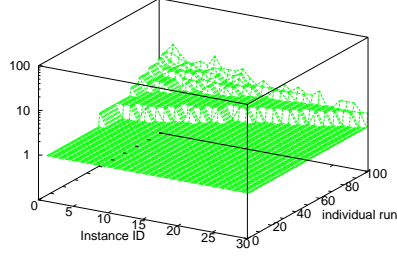
(e) Instance Class 5



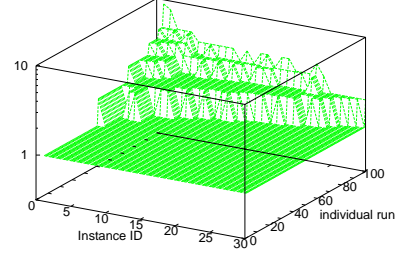
(f) Instance Class 6

Figure 5.21: The distributions of the iteration-to-target-value by 100 individual runs of TV to the 30 MAX-3-SAT instances from each of classes 1,2,3,4,5 and 6. A point (x, y, z) specifies an instance number x , a number of individual runs y and a number of iteration z and describes that y -th run of TV to the instance x , the z iterations were needed to find a solution with value at least as good as the target-value.

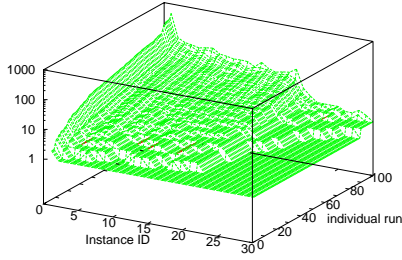
corresponding number of iterations. As we can see from the figures, the iteration-to-target-value is distributed differently in each class. For the instances of the classes 5, 7 and 8, the REF very quickly achieved the solutions found by the LPcut. In contrast, for the instances of the classes 2, 3, 4 and 12 the required numbers of iterations are quite high.



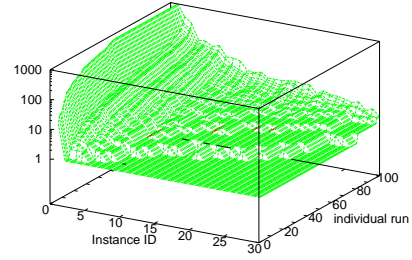
(a) Instance Class 7



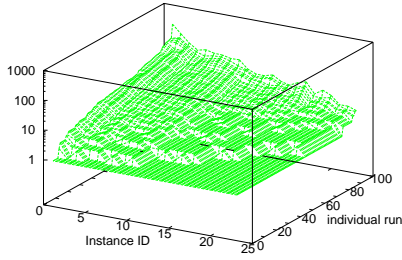
(b) Instance Class 8



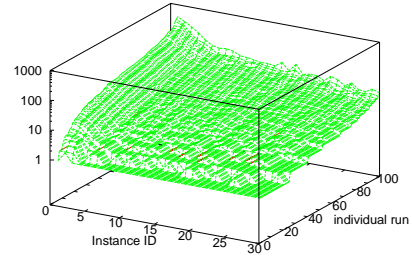
(c) Instance Class 9



(d) Instance Class 10



(e) Instance Class 11



(f) Instance Class 12

Figure 5.22: The distribution of the iteration-to-target-value by 100 individual runs of TV to the 30 MAX-3-SAT instances from each of classes 7,8,9,10,11 and 12 (note that 11th class consists only 23 graphs). A point (x, y, z) specifies an instance number x , a number of individual runs y and a number of iteration z and describes that y -th run of TV to the instance x , the z iterations were needed to find a solution with value at least as good as the target-value.

5.3.4. Max-4-SAT

In this section we describe the experimental results of our technique for the MAX-4-SAT problem.

Test Instances

We considered six classes of test instances in the experimental study of the application of our technique to the MAX-4-SAT problem. Table 5.10 gives an overview of the characteristics of each of these classes. The instances of the classes 1 to 3 have been generated by the **GTgraph** and the instances of the classes 4 to 6 have been generated by the Selman's generator. All instances are non-weighted (i.e., the weights of clauses are uniform).

class ID	number of instances	number or range of variables	number or range of clauses	ratio of number of clauses and number of variables
1	500	[500–1500]	[629–5621]	–
2	350	[500–1200]	[7545–43164]	–
3	500	[500–2500]	[2268–11250]	4.50
4	500	[500–1000]	[1138–2266]	2.26
5	500	[500–1000]	[1640–3266]	3.26
6	500	[500–1000]	[2140–4266]	4.26

Table 5.10: Various classes of test instances for the MAX-4-SAT.

Experimental Evaluation

The experiment consists of six parts, each part is considering one class of test instances. The Figures from 5.23 to 5.28 give the details of the computational results for all six classes of test instances: the ratio (p.c.) of start and final solutions' values from the local search in the **LPcut** procedure; the ratio (p.c.) of start solution value from the local search in the **LPcut** procedure and the solution value found by the **REF** procedure; and the ratio (p.c.) of solution values found by the **LPcut** and by the **REF**. Recall, that the algorithm **LPcut** runs the local search starting from the optimal solution found by **LP** (4.3).

The ratio of start solution value from the local search in the **LPcut** and the solution value found by the **REF** is no more than 5.6 p.c., in all instances of test classes. In the majority of instances (90 p.c.) this ratio is less than 5.0 p.c. From Figures 5.23 to 5.28 we observe that for the 4th, 5th and 6th classes of test instances the ratio of objective values

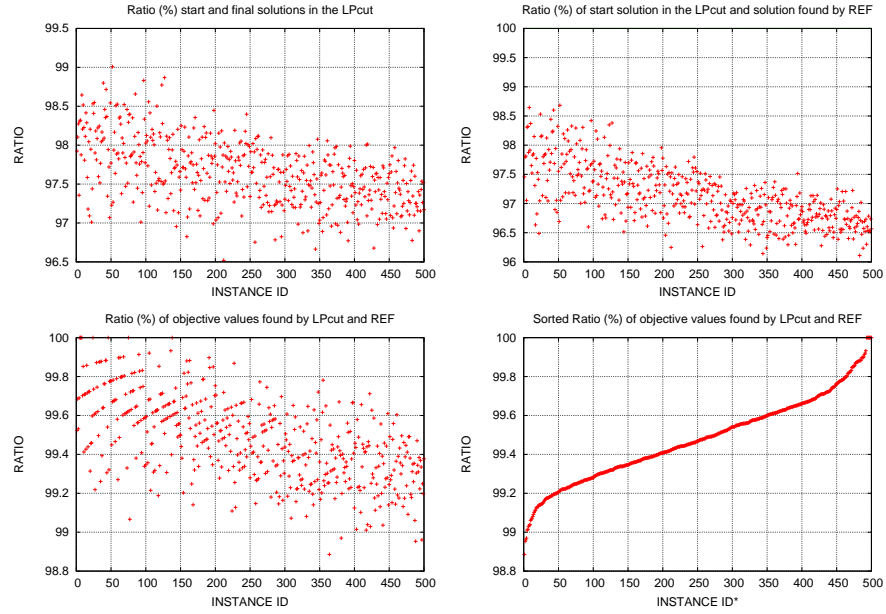


Figure 5.23: MAX-4-SAT test class 1: comparison of the solutions found by the LPcut and REF; comparisons of start solution delivered by the LP to each of the solutions found by the LPcut and by REF.

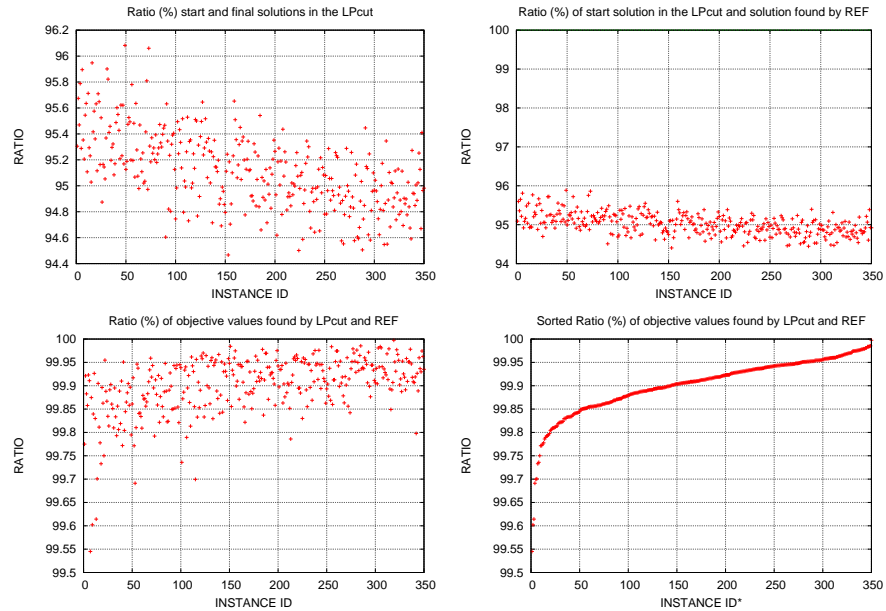


Figure 5.24: MAX-4-SAT test class 2: comparison of the solutions found by the LPcut and REF; comparisons of start solution delivered by the LP to each of the solutions found by the LPcut and by REF.

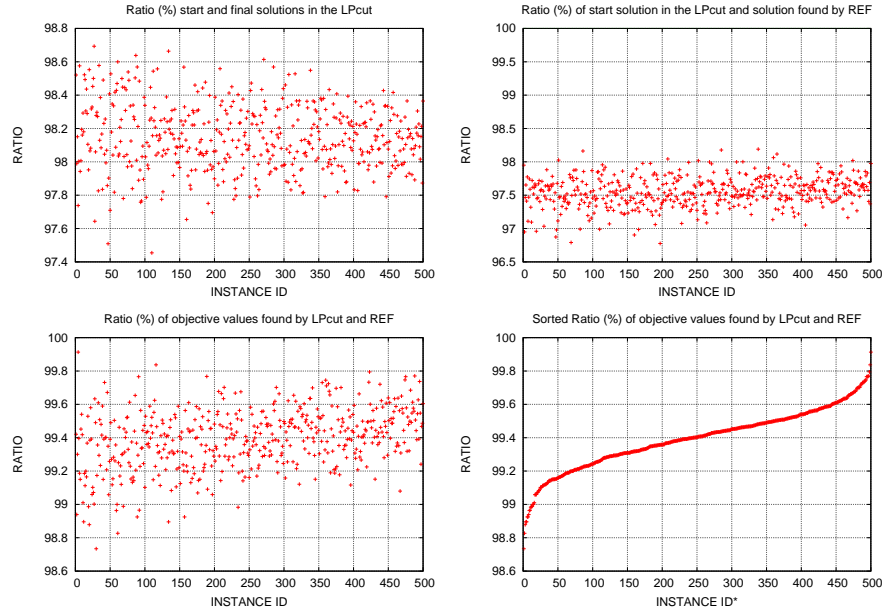


Figure 5.25: MAX-4-SAT test class 3: comparison of the solutions found by the LPcut and REF; comparisons of start solution delivered by the LP to each of the solutions found by the LPcut and by REF.

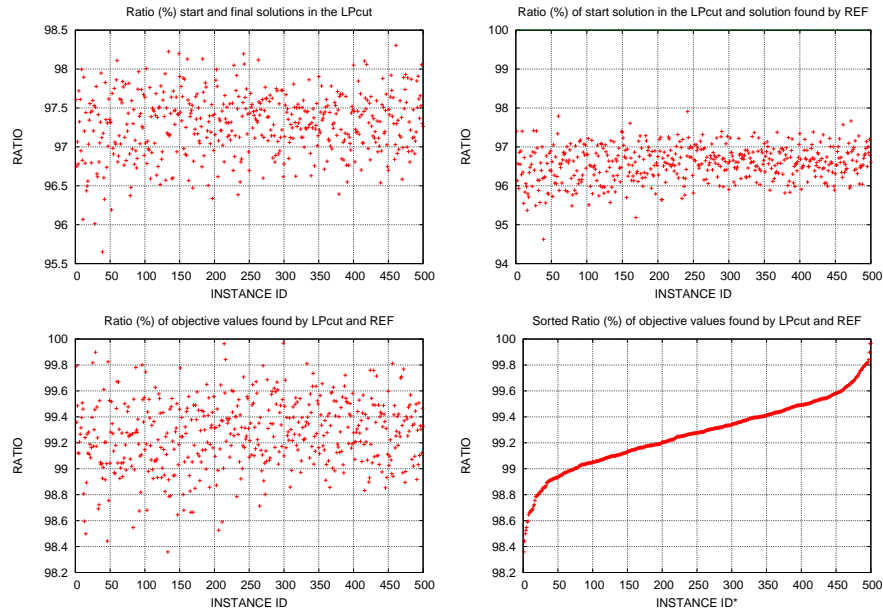


Figure 5.26: MAX-4-SAT test class 4: comparison of the solutions found by the LPcut and REF; comparisons of start solution delivered by the LP to each of the solutions found by the LPcut and by REF.

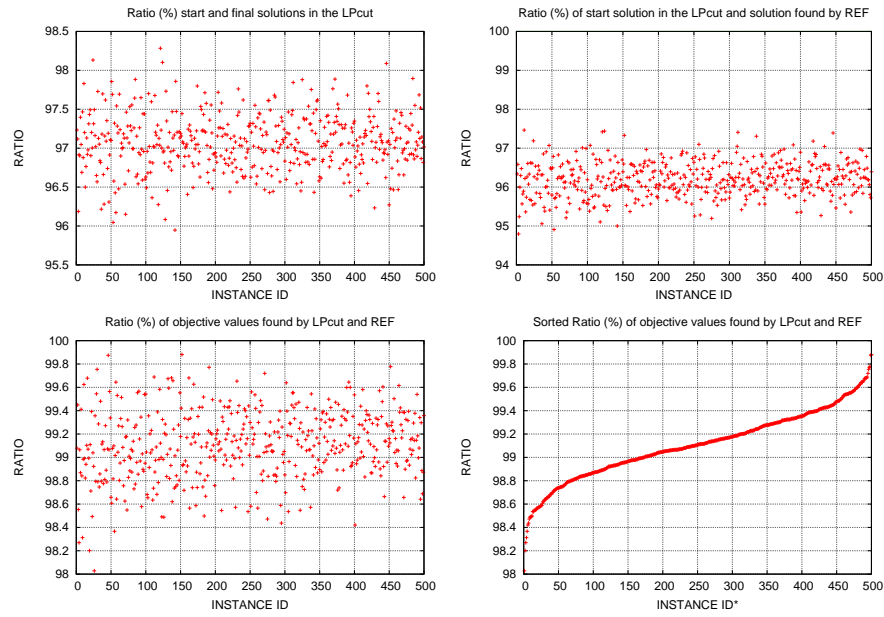


Figure 5.27: MAX-4-SAT test class 5: comparison of the solutions found by the LPcut and REF; comparisons of start solution delivered by the LP to each of the solutions found by the LPcut and by REF.

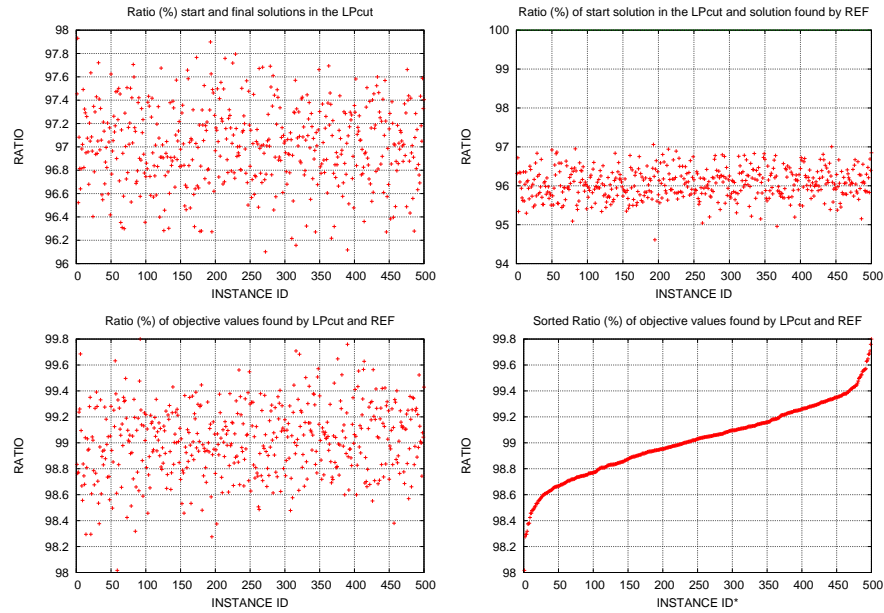


Figure 5.28: MAX-4-SAT test class 6: comparison of the solutions found by the LPcut and REF; comparisons of start solution delivered by the LP to each of the solutions found by the LPcut and by REF.

ID	Comparison of objective values					Comparison of CPU times		
	the least	90% \geq	average	the most	percentage	the most	90% \leq	average
(A)	(B)	(C)	(D)	(E)	(F)	(G)	(H)	(I)
1	98.88	99.21	99.48	100.00	1.00	0.0582	0.0053	0.0045
2	99.54	99.83	99.90	99.99	0.00	0.0038	0.0033	0.0029
3	98.73	99.16	99.39	99.91	0.00	0.0052	0.0042	0.0038
4	98.36	98.94	99.26	99.96	0.00	0.0117	0.0056	0.0053
5	98.03	98.74	99.10	99.88	0.00	0.0175	0.0065	0.0058
6	98.01	98.66	99.01	99.80	0.00	0.0134	0.0048	0.0043

Table 5.11: Comparison of our technique with the reference technique for the MAX-4-SAT problem. Columns: (A) ID of the experiment;

(B)–(E) ratios (p.c.) of the objective values found by LPcut and by REF: (B) the worst, (C) for 90 p.c. of all instances the ratio (p.c.) hits or exceeds this value, (D) the average, (E) the best;

(F) the percentage of instances where the outcome of LPcut was better than or equal to the outcome of the REF;

(G)–(I) ratios of the CPU times needed by LPcut and the REF: (G) the highest; (H) for 90 p.c. of all instances the ratio matches or does not exceed this value; (I) the average.

found by the LPcut and by the REF is slightly decreasing while the ratio of number of clauses and number of variables is increasing from the 4th to the 5th, and from the 5th to the 6th classes.

We summarize the computational results that are the ratios (p.c.) of the objective values found by LPcut and by REF, and the ratios of the CPU times needed by LPcut and by REF in Table 5.11. The worst case ratio (Column (B)) of objective values found by LPcut and by REF is less than 2.0 p.c., and the ratio for 90 p.c. quantile (Column (C)) is less than 1.5 p.c. in all instance classes.

5.4. The Longest Path with Source and Sink

In this section we present a quite interesting experimental results of our technique for the LONGEST DIRECTED PATH problem. We describe the generation of test instances, and then present the computational results.

5.4.1. Generation of Instances

In our experimental evaluation we consider 13 classes of test instances. The instances of the classes from 1 to 9 are generated by rudy, and the instances of the remaining 4 classes

are generated by GTgraph.

class ID	number of instances	number of nodes	density	arc weight	generator type
1	1000	[500–1500]	5.00	uniform	rudyl : <i>simple-random</i>
2	1000	[500–1500]	5.00	uniform distr. of $[1, \dots, 100]$	rudyl : <i>simple-random</i> ^a
3	1000	[500–1500]	5.00	uniform distr. of $[1, \dots, 10000]$	rudyl : <i>simple-random</i> ^a
4	1000	[500–1500]	0.50	uniform	rudyl : <i>simple-random</i>
5	1000	[500–1500]	0.50	uniform distr. of $[1, \dots, 100]$	rudyl : <i>simple-random</i> ^b
6	1000	[500–1500]	0.50	uniform distr. of $[1, \dots, 100000]$	rudyl : <i>simple-random</i> ^b
7	1000	2000	1.22	uniform distr. of $[1, \dots, 100]$	rudyl : <i>almost-planar</i>
8	1000	2000	1.22	uniform distr. of $[1, \dots, 100000]$	rudyl : <i>almost-planar</i> ^c
9	1000	1000	[0.37–15.27]	uniform	rudyl : <i>simple-random</i>
10	1000	[500–1500]	[0.60–0.77]	uniform	GTgraph : <i>Erdős-Renyi</i>
11	1000	[500–1500]	[0.60–0.77]	uniform distr. of $[1, \dots, 10000]$	GTgraph : <i>Erdős-Renyi</i> ^d
12	1000	[500–1500]	0.65	uniform	GTgraph : <i>random</i>
13	1000	[500–1500]	0.65	uniform distr. of $[1, \dots, 10000]$	GTgraph : <i>random</i> ^e

^athis class is similar to the 1st class, but graphs differ by the arc weights.

^bthis class is similar to the 4th class, but graphs differ by the arc weights.

^cthis class is similar to the 7th class, but graphs differ by the arc weights.

^dthis class is similar to the 10th class, but graphs differ by the arc weights.

^ethis class is similar to the 12th class, but graphs differ by the arc weights.

Table 5.12: Various classes of test instances for the DIRECTED LONGEST PATH.

Table 5.12 summarizes the characteristics of each class: the number of instances in that class, (the range of) the number of nodes, (the range of) the density, and the construction of arc weights of instances in that class, and which generator has been used to generate these instances.

5.4.2. Summary of Computational Results

Similarly to the experimental evaluations for the MAX-DI-CUT problem with source and sink, and the variations of MAX- k -SAT, $k \geq 2$, we compared our technique for the LONGEST DIRECTED PATH with the reference technique, which repeatedly starts local search at random start solutions. We experimented with two distinct methods to generate random start solutions in the reference technique: a random depth first search (DFS) starts at source node, and an adaptation of a Pohl-Warnsdorf rule [123] for generating long paths in undirected graph. We describe these two methods.

Random DFS Starting from a source node, s , we simply select the next node at random that does not lead to a circle. When we reach a sink node, t , we are done. We denote

Algorithm 7 Start Solution Generator with Random DFS

```

procedure Start-DFS
  set  $P \leftarrow \emptyset$ ;
   $\forall v \in V \setminus \{s\}$  set  $Color(v) \leftarrow white$ ;
  if Random-DFS( $s$ ) = true then
    return  $P$ ;
  end if
return false;

```

the procedure as **Start-DFS**. A pseudocode of this method of generating the random start solutions is shown in Algorithm 7. The **Start-DFS** calls the recursive procedure **Random-DFS**, whose pseudocode is given in Algorithm 8. For a given directed graph $G = (V, A)$ with nonnegative weight function $c : A \rightarrow \mathcal{R}_0^+$ and source and sink, $s, t \in V$, the procedure **Start-DFS** generates a random $s - t$ path P if there exists at least one.

Pohl-Warnsdorf Method The Pohl-Warnsdorf rule [123, 124] is a greedy heuristic that tries to find the longest path in a graph. It was based on the work of the 19th century mathematician Warnsdorf, who proposed a rule for finding Knights tours on a chess board: “Go to a next square which has the fewest ways out.” (A Knights tour on the chess board is a closed tour with Knights move from square to square.) Pohl [123] modified the rule to be recursive in tie-breaking situations, and generalized for finding an arbitrary Hamiltonian path in a graph, and proposed a Pohl-Warnsdorf algorithm (PW): “go to the next node of least degree”. This algorithm was very effective to the Knights tour problem and to

the Tutte's graph [152]. We adapted the Pohl-Warnsdorf algorithm to generate the start solutions, the $s - t$ paths, for the reference technique. We denote this method of start solution generation as **Start-PW**, and describe the pseudocode in Algorithm 9.

Algorithm 8 Random DFS

```

procedure Random-DFS( $v$ )
  if  $Color(v) \neq white$  then
    return  $false$ ;
  end if
  set  $Color(v) \leftarrow grey$ ,  $P \leftarrow P \cup \{v\}$ ;
  set  $M \leftarrow \{w \in V : \exists (v, w) \in A, w \notin P\}$ ; (this is a set of successor nodes)
  if  $M = \emptyset$  then
    return  $false$ ;
  end if
  repeat
    choose randomly  $u \in M$ ;
    if  $u = t$  then
       $P \leftarrow P \cup \{t\}$ 
      return  $true$ ;
    end if
    if Random-DFS( $u$ ) =  $true$  then
      return  $true$ 
    else
      backtrack: remove  $u$  from  $M$  and  $P$ ;
    end if
  until  $M = \emptyset$ ;

```

Algorithm 9 Start Solution Generator with Pohl-Warnsdorf Algorithm

```

procedure Start-PW
  set  $P \leftarrow \emptyset$ 
   $\forall v \in V \setminus \{s\}$  set  $Color(v) \leftarrow white$ ;
  if PW( $s$ ) =  $true$  then
    return  $P$ ;
  end if
  return  $false$ ;

```

The **Start-PW** algorithm initializes an $s - t$ path P as the empty path and the colors of nodes to white. A color of a node indicates whether the node is visited: white for not visited and grey for visited. By starting at the source node, s , the algorithm calls the recursive procedure PW, the adapted Pohl-Warnsdorf rule. The return value $true$ of PW indicates the

Algorithm 10 The modified Pohl-Warnsdorf Algorithm for the $s - t$ path

```

procedure PW( $v$ )
  if  $Color(v) \neq white$  then
    return  $false$ ;
  end if
  set  $Color(v) \leftarrow grey$ ,  $P \leftarrow P \cup \{v\}$ ;
  if  $v = t$  then
    return  $true$ ;
  end if
  set  $M \leftarrow \{w \in V : \exists(v, w) \in A, w \notin P\}$ ; (this is a set of successor nodes)
  while  $M \neq \emptyset$  do
    select  $u \in M$  with least degree:  $d_u = \min_{w \in M} d_w$ ;
    if  $u$  is unique then
       $v^* \leftarrow u$ ;
    else
      set  $M' \leftarrow \{r \in M : d_r = d_u\}$ 
      let  $N(r) = \{w \in V : \exists(r, w) \in A, w \notin P\}$ ,  $\forall r \in M'$ ;
      let  $d'(r) = \min_{w \in N(r)} |\{h \in V : \exists(w, h) \in A, h \notin P\}|$ ,  $\forall r \in M'$ ;
      select  $r' \in M'$  with successor node of least degree  $d'(r')$ :  $d'(r') = \min_{r \in M'} d'(r)$ ;
      if  $r'$  is unique then
         $v^* \leftarrow r'$ ;
      else
        choose randomly  $r \in M'$ ;
         $v^* \leftarrow r$ ;
      end if
    end if
    if PW( $v^*$ ) =  $true$  then
      return  $true$ 
    else
      backtrack: remove  $v^*$  from  $M$  and  $P$ ;
    end if
  end while
  return  $false$ ;

```

successful construction of $s - t$ path P .

The PW procedure goes as follows. The current node is given. The current node is placed on the path, then the node becomes grey. From the current node we want to move to the adjacent (successor) node that has the least degree. If such node is unique (its degree is less than the degrees of other successor nodes), then we select this node and call the recursion (it becomes the new current node). Otherwise (if there are two or more nodes with equal least degrees), we want to break the tie by examining the degrees of adjacent nodes (successor nodes) of these “equaled” nodes. Let us denote these “equaled” nodes as *candidate* nodes. Here, to count the degrees of candidate nodes we exclude the incident arcs connecting the nodes that are on the path. If there is a unique candidate node that has a successor node with least degree, then we select that candidate node; we call the recursion. Otherwise, we select an arbitrary node from the candidate nodes; we call the recursion. On a formation of any cycle we backtrack. We are done when the sink, node t , is reached.

Preliminary experiments (which are not reported in this thesis) with the reference technique that uses one of the two start solution generators have shown that the REF with Random-DFS and the REF with Start-PW have found different results in same problem instances. Therefore, we use both of Random-DFS and Start-PW methods in the generation of start solutions in the reference technique.

We modified the reference technique such that in each of its iteration, first, we choose one of the two start solution generators, Start-DFS or Start-PW, at random; then, we start the local search at the generated start solution. We denote this modified reference technique as REF2. Its pseudocode is shown in Algorithm 11.

Algorithm 11 Modified Reference Strategy

```

procedure REF2( $k_{max}$ )
  for  $k = 1, \dots, k_{max}$  do
    choose one of start path generators at random: Start-DFS or Start-PW;
    generate a random path  $P_0$  by the chosen generator;
     $P' \leftarrow \text{LocalSearchLP}(P_0)$ ;
    if  $k = 1$  then
       $P^* \leftarrow P'$ ;
    else if  $C(P^*) < C(P')$  then
       $P^* \leftarrow P'$ ;
    end if
  end for
  return  $P^*$ ;

```

ID	Comparison of objective values					Comparison of CPU times		
	the least	90% \geq	average	the most	percentage	the most	90% \leq	average
(A)	(B)	(C)	(D)	(E)	(F)	(G)	(H)	(I)
1	96.60	98.05	99.03	100.20	4.00	0.0065	0.0055	0.0045
2	97.04	99.97	104.58	114.44	89.00	0.0068	0.0057	0.0047
3	97.85	99.82	103.66	114.39	87.00	0.0052	0.0043	0.0037
4	86.51	100.00	111.59	148.40	89.00	0.0071	0.0054	0.0044
5	93.44	101.86	116.50	140.18	93.00	0.0051	0.0044	0.0038
6	95.09	101.17	115.75	141.49	92.00	0.0049	0.0043	0.0037
7	93.96	101.15	115.54	136.12	93.00	0.0070	0.0051	0.0046
8	95.17	101.11	114.00	134.88	93.00	0.0064	0.0050	0.0045
9	89.95	97.38	100.35	133.33	21.00	0.0062	0.0043	0.0031
10	86.17	105.98	117.02	136.42	95.00	0.0760	0.0065	0.0053
11	92.27	99.25	113.88	136.05	88.00	0.0725	0.0056	0.0063
12	92.22	94.88	97.88	104.15	21.00	0.0092	0.0052	0.0040
13	96.83	100.21	107.98	123.30	91.00	0.0108	0.0041	0.0034

Table 5.13: Comparison of our technique with the reference technique for the LONGEST DIRECTED PATH. Columns: (A) ID of the experiment; (B)–(E) ratios (p.c.) of the objective values found by LPcut and by REF2: (B) the worst, (C) for 90 p.c. of all instances the ratio (p.c.) hits or exceeds this value, (D) the average, (E) the best; (F) the percentage of instances where the outcome of LPcut was better than or equal to the outcome of the REF2; (G)–(I) ratios of the CPU times needed by LPcut and the REF2: (G) the most; (H) for 90 p.c. of all instances the ratio does not exceed this value; (I) the average.

Computational Results The experiment for the LONGEST DIRECTED PATH consists of thirteen parts; each part considered one class of test instances. We summarize the computational results – the ratios of objective values found by LPcut and by REF2, and the ratios of CPU times needed by LPcut and by REF2 – in a Table 5.13. Owing to the extensive running time we tested only 575 instances of the 9th test class.

In the worst case (column (B)) of all test instances our technique was 14 p.c. (in experiment with an ID of 10) worse than the reference technique. But in all classes, the 90 p.c. quantile (column (C)) were nearly 6 p.c. or smaller. On average, our technique found longer paths than reference technique to all test instance classes, except the 1st and 12th classes. The average ratios (column (D)) are less in the test classes that consist of unweighted graphs (1st, 9th and 12th classes) than in the test classes which consist of weighted graphs (2nd, 3rd, 5th, 6th and 13th classes).

Let us take a closer look at the computational results of the 1st, 4th and 9th classes of instances, which consist of uniformly weighted graphs. Remember that, the first class

contains the (dense) graphs with the density of 5.0 p.c., the fourth class contains the (sparse) graphs with the density of 0.5 p.c., and the ninth class consists of graphs with various densities from 0.5 p.c. to 12.0 p.c. (but the tested 575 instances vary in density from 0.5 p.c. to 8.91 p.c.). Besides of that comparing the two heuristics with one another, we compared the results of our technique and the reference technique with the trivial upper bound – Hamiltonian path. The Figures 5.29, 5.30 and 5.31 display the comparisons of our technique with the reference technique for the first, fourth and ninth classes – the ratios of path-lengths found by our technique and the corresponding upper bounds, and the ratios of path-lengths found by reference technique and the corresponding upper bounds, respectively.

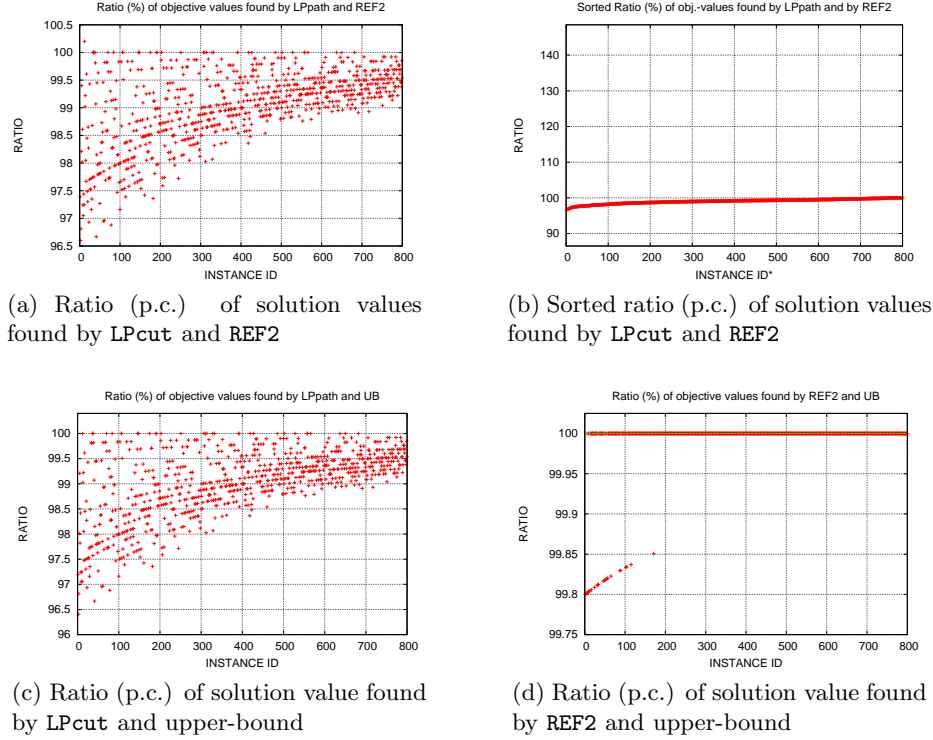


Figure 5.29: The comparison of path-lengths found by our technique and by reference technique to the **first** class of the LONGEST DIRECTED PATH instances¹, and the comparison of them with the trivial upper bound.

1st class: In graphs of the first class the LPcut found on average 0.97 p.c. shorter paths than the REF2. On average, the LPcut found path with length of 99.02 p.c. length of

¹ID of instances in Figure 5.29b are sorted through the ratios

Hamiltonian path. On average the REF2 found path with length of 99.99 p.c. length of Hamiltonian path. In 96.6 p.c. of test instances the REF found Hamiltonian paths (optimal solutions).

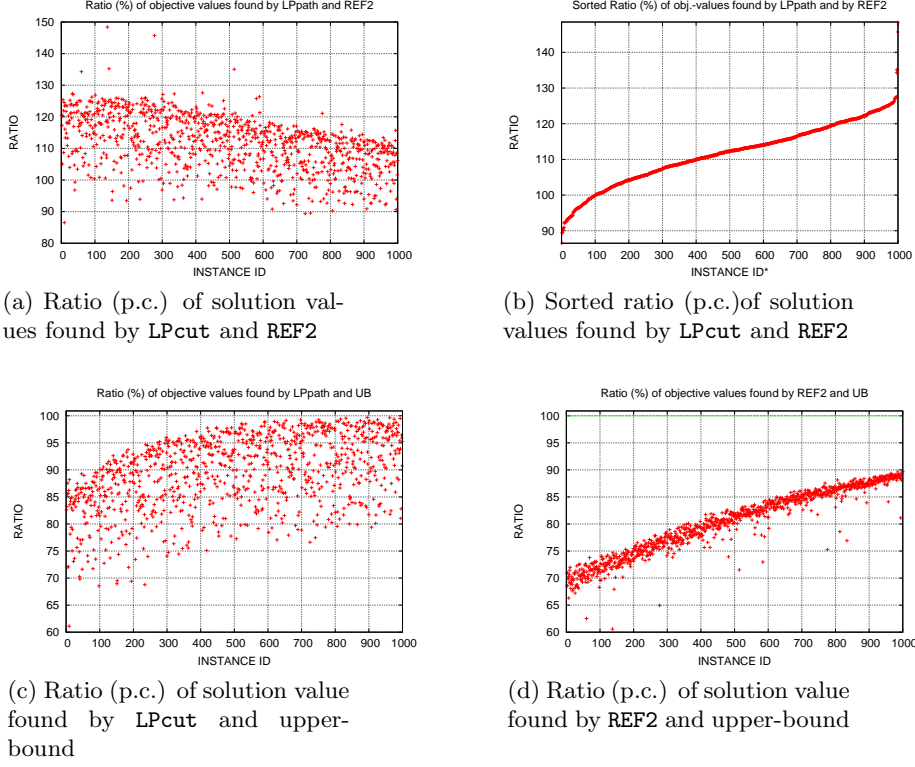


Figure 5.30: The comparison of path-lengths found by our technique and by reference technique to the **fourth** class of the LONGEST DIRECTED PATH instances ¹, and the comparison of them with the trivial upper bound.

4th class: In graphs of the fourth class the LPcut found 11.6 p.c. longer paths than the REF2, on average. The LPcut found the path with length of 89 p.c. of Hamiltonian path, and REF2 found path with length of 80 p.c. of Hamiltonian path, on average. For about 31 p.c. of the instances LPcut found the paths with lengths of at least 95 p.c. of lengths of Hamiltonian paths. But the REF2 found paths with lengths of at most 89 p.c. of lengths of Hamiltonian paths. Yet, the existence of Hamiltonian path for these (sparse) graphs is not known.

9th class: In the experimental on instances of the ninth class the LPcut found paths with length of almost equal to the REF2. In graphs with density equal to or more than 2.07

¹ID of instances in Figure 5.30b are sorted through the ratios

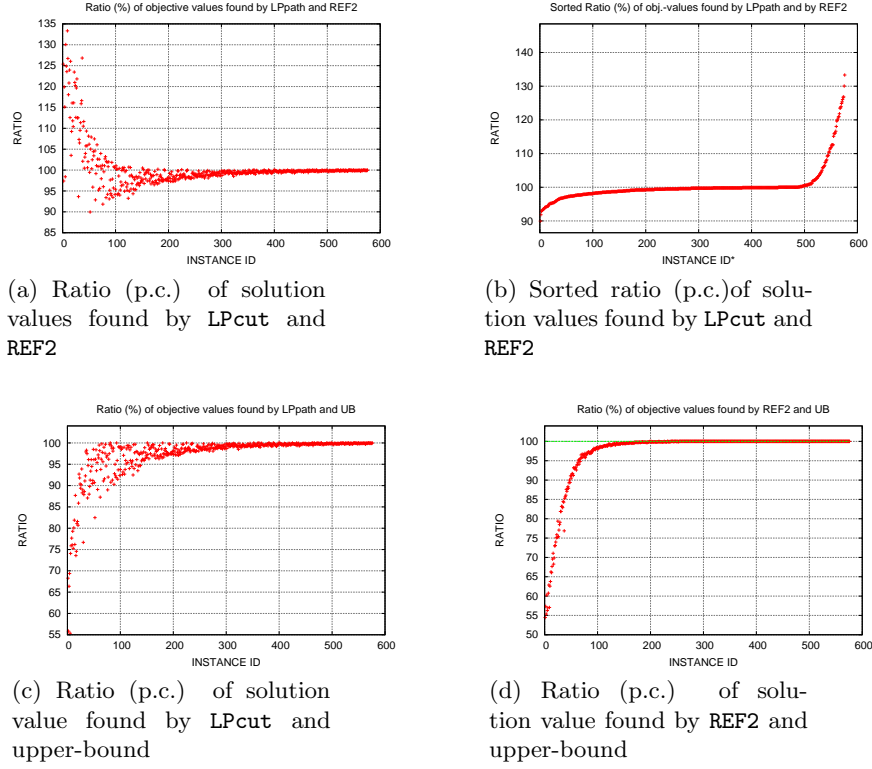


Figure 5.31: The comparison of path-lengths found by our technique and by reference technique to the **ninth** class of the LONGEST DIRECTED PATH instances ¹, and the comparison of them with the trivial upper bound.

p.c. the REF2 found paths with length at least of 99.99 p.c. of Hamiltonian path, and in instances where the density is more than 4.06 p.c. the REF2 found Hamiltonian paths – the optima. These optimally solved instances make 56.08 p.c. of tested instances. For these instances the LPcut found the objective values at least 98.1 p.c. of optimal values. LPcut found Hamiltonian paths for only 35 instances (6.08 p.c. of tested instances).

The experimental results are not delivering the fact of that both techniques have been found nearly optimal solutions. For some instance classes (1st and 12th) two algorithms found paths whose lengths do not differ much from each other. On the other hand, there are strong mavericks, whose lengths differ far away as 40 p.c. These experimental results are supporting our claim of that the strong coincidences of substantially different techniques to the MAX-DI-CUT and the MAX- k -SAT problems, where $k \in \{2, 3, 4\}$, not happened by a

¹ID of instances in Figure 5.31b are sorted through the ratios

chance.

The computational results are showing that our technique found longer paths (at least 3.5 p.c.) than the reference technique to the weighted graphs, on average. Recall that, there exists no constant factor approximation algorithm for the LONGEST PATH unless $\mathcal{P} = \mathcal{NP}$ [97].

Even though we do not conclude that our technique finds near optimal solutions to the LONGEST DIRECTED PATH, as the computational study suggests, yet as a consequence of its small running time, our technique can be used to find promising lower bound for the LONGEST DIRECTED PATH problem with source and sink.

Chapter 6

Conclusion

We presented hybrid LP-based method to generate good start solutions for local search. We implemented our technique for five exemplary optimization problems, the MAX-DI-CUT with a source and a sink, three variations of the MAX- k -SAT problem, where $k = 2$, $k = 3$ and $k = 4$, and the LONGEST DIRECTED PATH with source and sink. It is a requirement to apply our method that there exists an LP such that the optimal basis solutions to the LP are feasible to given optimization problem. This method could be applied to many optimization problems, but an adaptation to a given problem is not entirely trivial. We showed where it is promising to look for such an LP: the quadratic or nonlinear programming formulation of the problem. The MAX-DI-CUT and the MAX- k -SAT, $k \geq 2$, problems can be implemented as nonlinear programming. In case of the MAX-DI-CUT we substituted the quadratic objective function by a linear function and in case of the MAX- k -SAT, $k \geq 2$, we substituted the quadratic constraints by linear constraints to obtain the required LP. For the LONGEST DIRECTED PATH we implemented our technique in an alternative way, in that the related LP is not directly obtained from the nonlinear formulation of the problem, but some of the constraint equalities of the LP are originated from another optimization problem – network flow problem.

In the experimental study we compared our technique against the (multistart) reference technique: repeated local search starting from random solutions. Even though the two algorithms substantially differ from each other, the solution values found by our algorithm and by the reference algorithm differed marginally from each other throughout all the experiments with various classes of the MAX-DI-CUT and the MAX- k -SAT test instances. This continuous coincidence of the results of two different algorithms in various classes of

the instances suggests that both our algorithm and the reference algorithm have found permanently near-optimal solutions. The run time of our algorithm was rather small. The experiments, the evaluations of iteration-to-target-value, confirmed the quality of solution-time of our algorithm.

In a case of the MAX-2-SAT, the comparison of results of our algorithm with the theoretical lower and upper bounds to the instances, for which these bounds were available, supported our above mentioned conclusion. Indeed, the lower and upper bounds have no methodical relations with our algorithm. However, the comparison has shown that our empirical results are very tight to the theoretical bounds.

The experimental results to the LONGEST DIRECTED PATH problem with source and sink did not let us draw a conclusion of that two algorithms have found near-optimal solutions. In the computational study to the various classes of the LONGEST DIRECTED PATH instances, two algorithms found solutions of nearly identical as well as highly variable values. The occurrences of a number of mavericks in the experimental study support our conclusion, that our algorithm has found near-optimal solutions to these problems, which we have drawn from our empirical results to the MAX-DI-CUT and the MAX- k -SAT, $k \in \{2, 3, 4\}$. Furthermore, the small run time of our algorithm and the experimental study did encourage of the using our technique to obtain good lower bound to the LONGEST DIRECTED PATH problem.

The simplicity and the small computing time of our technique is a motivation to study possible adaptations of our technique to further hard optimization problems. As we mentioned above, the requirement of applying our technique to a given optimization problem is an existence of an LP, of which optimal basis solutions are feasible to optimization problem. Therefore, in each case of the adaptations of our technique to optimization problems, the underlying problem should be carefully examined whether there exists required LP. For example, for a given problem, one could look into a nonlinear formulation of the problem, then substitute the nonlinear ingredients by linear ingredients to obtain the suitable LP. If we are given a problem that we could formulate as an optimization problem of finding extreme points of nonlinear function over some integral polyhedron (as the cases for many combinatorial optimization problems), then we would try to approximate the nonlinear objective as linear objective and optimize the hyperplane over the integral polyhedron. This could lead to a point, where it is promising to start the search for an optimal solution.

Currently the interest in combining LP and metaheuristics to tackle the hard combinatorial optimization problems is increasing. Our method contributes to the view that the delicate combinations of exact and metaheuristic algorithms deliver promising methods for hard optimization problems.

List of Tables

5.1. Various classes of test instances for the MAX-DI-CUT problem	72
5.2. Comparison of our technique with the reference technique for MAX-DI-CUT problem	74
5.3. Comparison (ratio (p.c.) of values) of the start and final solutions from local search in the LPcut procedure to the MAX-DI-CUT instances.	77
5.4. The various classes of test instances for the MAX-2-SAT problem	86
5.5. Comparison of our technique with the reference technique for the MAX-2-SAT problem.	88
5.6. Comparison (ratio (p.c.) of values) of the start and final solutions from local search in the LPcut procedure to the MAX-2-SAT instances.	89
5.7. Various classes of test instances for the MAX-3-SAT problem	97
5.8. Comparison of our technique with the reference technique for the MAX-3-SAT problem.	98
5.9. Comparison (ratio (p.c.) of values) of the start and final solutions from local search in the LPcut procedure to the MAX-3-SAT instances.	99
5.10. Various classes of test instances for the MAX-4-SAT.	102
5.11. Comparison of our technique with the reference technique for the MAX-4-SAT problem.	106
5.12. Various classes of test instances for the DIRECTED LONGEST PATH.	107
5.13. Comparison of our technique with the reference technique for the LONGEST DIRECTED PATH.	112
B.1. The computational results of the two algorithms to the MAX-2-SAT instances of the second class (Section 5.3.2): the ratio (p.c.) of the solution values found by LPcut and REF; the ratio of the CPU times needed by LPcut and REF.	146

List of Figures

5.1. Ratio of objective values found by LPcut and by REF to the MAX-DI-CUT instances of the classes 1 to 4.	75
5.2. Ratio of objective values found by LPcut and by REF to the MAX-DI-CUT instances of the classes 5 to 8.	76
5.3. Ratio of objective values found by LPcut and by REF to the MAX-DI-CUT instances of the classes 9 to 12.	76
5.4. The distributions of the iteration-to-target-value in 100 individual runs of TV to 54 MAX-DI-CUT instances of the eleventh class.	79
5.5. The map of the distributions of the iteration-to-target-value shown in Figure 5.4.	79
5.6. The distribution of the iteration-to-target-value by 100 runs of REF to all 54 instances of the eleventh class of MAX-DI-CUT.	80
5.7. The distributions of the iteration-to-target-value by 100 runs of TV to 32 MAX-DI-CUT instances of the first class.	81
5.8. The distribution of the iteration-to-target-value by 100 runs of REF to 32 instances of the first class of MAX-DI-CUT.	81
5.9. The distributions of the iteration-to-target-value by the 100 individual runs of TV to all instances of the sixth class of MAX-DI-CUT.	82
5.10. The map of the distributions of the iteration-to-target-value shown in Figure 5.9.	83
5.11. The ratio of obj. values found to the MAX-2-SAT instances	87
5.12. The distributions of the iteration-to-target-value in 200 individual runs of TV to the instances 3, 11 and 16 of the second class of MAX-2-SAT.	90
5.13. The distributions of the iteration-to-target-value in 100 individual runs of TV to the instances 22 and 23 of the second class of MAX-2-SAT.	90
5.14. The distributions of the iteration-to-target-value by 60 individual runs of TV to 50 MAX-2-SAT instances of the 4th class.	91
5.15. The map of the distributions of the iteration-to-target-value, which are shown in Figure 5.14.	91
5.16. The distribution of the iteration-to-target-value in 60 runs of the TV to all 50 MAX-2-SAT instances of the 3rd class.	92

5.17. The distributions of the iteration-to-target-value by 60 individual runs of TV to 50 MAX-2-SAT instances of the third class.	93
5.18. The map of the distributions of the iteration-to-target-value, which are shown in Figure 5.17.	93
5.19. The ratio (p.c.) of objective value found by the LPcut and theoretical lower bound, and the ratio of objective value found by the LPcut and theoretical upper bound . . .	95
5.20. The ratio (p.c.) of objective value found by the REF and theoretical lower bound, and the ratio of objective value found by the REF and theoretical upper bound . . .	95
5.21. The distributions of the iteration-to-target-value by 100 individual runs of TV to the 30 MAX-3-SAT instances from each of classes 1,2,3,4,5 and 6. . . .	100
5.22. The distribution of the iteration-to-target-value by 100 individual runs of TV to the 30 MAX-3-SAT instances from each of classes 7,8,9,10,11 and 12 . . .	101
5.23. MAX-4-SAT test class 1: comparison of the solutions found by the LPcut and REF; comparisons of start solution delivered by the LP to each of the solutions found by the LPcut and by REF.	103
5.24. MAX-4-SAT test class 2: comparison of the solutions found by the LPcut and REF; comparisons of start solution delivered by the LP to each of the solutions found by the LPcut and by REF.	103
5.25. MAX-4-SAT test class 3: comparison of the solutions found by the LPcut and REF; comparisons of start solution delivered by the LP to each of the solutions found by the LPcut and by REF.	104
5.26. MAX-4-SAT test class 4: comparison of the solutions found by the LPcut and REF; comparisons of start solution delivered by the LP to each of the solutions found by the LPcut and by REF.	104
5.27. MAX-4-SAT test class 5: comparison of the solutions found by the LPcut and REF; comparisons of start solution delivered by the LP to each of the solutions found by the LPcut and by REF.	105
5.28. MAX-4-SAT test class 6: comparison of the solutions found by the LPcut and REF; comparisons of start solution delivered by the LP to each of the solutions found by the LPcut and by REF.	105
5.29. The comparison of path-lengths found by our technique and by reference technique to the first class of the LONGEST DIRECTED PATH instances . . .	113
5.30. The comparison of path-lengths found by our technique and by reference technique to the first class of the LONGEST DIRECTED PATH instances . . .	114
5.31. The comparison of path-lengths found by our technique and by reference technique to the first class of the LONGEST DIRECTED PATH instances . . .	115
B.1. The distribution of the iteration-to-target-value by 100 individual runs of the TV to the instances with ID numbers of 1 to 3 of the eleventh class for the MAX-DI-CUT.	141
B.2. The distribution of the iteration-to-target-value by 100 individual runs of the TV to the instances with ID numbers from 4 to 18 of the eleventh test class for the MAX-DI-CUT.	142

B.3. The distribution of the iteration-to-target-value for 100 individual runs of the TV to the instances with ID numbers of 19 to 33 of the eleventh test class for the MAX-DI-CUT.	143
B.4. The distribution of the iteration-to-target-value by 100 individual runs of the TV to the instances with ID numbers of 34 to 48 of the eleventh test class for the MAX-DI-CUT.	144
B.5. The distribution of the iteration-to-target-value by 100 individual runs of the TV to the instances with ID numbers of 49 to 54 of the eleventh test class of MAX-DI-CUT.	145
B.6. The distribution of the iteration-to-target-value by 60 runs of the TV to the instances with ID numbers of 0 to 11 of the fourth class of MAX-2-SAT instances.	147
B.7. The distribution of the iteration-to-target-value by 60 runs of the TV to the instances with ID numbers of 12 to 26 of the fourth class of MAX-2-SAT instances.	148
B.8. The distribution of the iteration-to-target-value by 60 runs of the TV to the instances with ID numbers of 27 to 41 of the fourth class of MAX-2-SAT instances.	149
B.9. The distribution of the iteration-to-target-value by 60 runs of the TV to the instances with ID numbers of 42 to 49 of the fourth class of MAX-2-SAT instances.	150

Bibliography

- [1] ALEX, R. M., RESENDE, M. G. C., AND RIBEIRO, C. C. Probability distribution of solution time in GRASP: An experimental investigation. *Journal of Heuristics* 8, 3 (2002), 343–373.
- [2] ALON, N., YUSTER, R., AND ZWICK, U. Color-coding: a new method for finding simple paths, cycles and other small subgraphs within large graphs. In *STOC '94: Proceedings of the twenty-sixth annual ACM symposium on Theory of computing* (New York, NY, USA, 1994), ACM Press, pp. 326–335.
- [3] ALSINET, T., MANYÀ, F., AND PLANES, J. An efficient solver for weighted Max-SAT. *Journal of Global Optimization* 41, 1 (2008), 61–73.
- [4] ALVAREZ-VALDES, R., PARAJON, A., AND TAMARIT, J. M. A computational study of LP-based heuristic algorithms for two-dimensional guillotine cutting stock problems. *OR Spectrum* 24, 2 (2002), 179–192.
- [5] ARORA, S., BOLLOBÁS, B., AND LOVÁSZ, L. Proving integrality gaps without knowing the linear program. In *FOCS '02: Proceedings of the 43rd Symposium on Foundations of Computer Science* (Washington, DC, USA, 2002), IEEE Computer Society, pp. 313–322.
- [6] ARORA, S., KARGER, D., AND KARPINSKI, M. Polynomial time approximation schemes for dense instances of NP-hard problems. In *STOC '95: Proceedings of the twenty-seventh annual ACM symposium on Theory of computing* (New York, NY, USA, 1995), ACM Press, pp. 284–293.
- [7] ARORA, S., LUND, C., MOTWANI, R., SUDAN, M., AND SZEGEDY, M. Proof verification and the hardness of approximation problems. *J. ACM* 45, 3 (1998), 501–555.
- [8] ARORA, S., AND SAFRA, S. Probabilistic checking of proofs: a new characterization of NP. *J. ACM* 45, 1 (1998), 70–122.
- [9] ASAHIRO, Y., IWAMA, K., AND MIYANO, E. Random generation of test instances with controlled attributes. In *Cliques, Coloring, and Satisfiability* (1996), D. S. Johnson and M. A. Trick, Eds., AMS, pp. 377–393.
- [10] AVDIL, A., AND WEIHE, K. Local search starting from an LP-solution – fast and quite good. submitted for publication to ACM, 2009.
- [11] BADER, D. A., AND MADDURI, K. *GTgraph: A Synthetic Graph Generator Suite*. Atlanta, GA, February 2006.
- [12] BALAS, E. Intersection cuts – A new type of cutting planes for integer programming. *Operations Research* 19 (1971), 19–39.
- [13] BALAS, E. Facets of knapsack polytope. *Mathematical Programming* 8, 1 (1975), 146–164.

- [14] BALAS, E. Disjunctive programming and a hierarchy of relaxations for discrete optimization problems. *SIAM Journal on Algebraic and Discrete Methods* 6, 3 (1985), 466–486.
- [15] BALAS, E. Disjunctive programming: Properties of the convex hull of feasible points. *Discrete Applied Mathematics* 89 (1998), 3–44.
- [16] BALAS, E., CERIA, S., AND CORNUÉJOLS, G. A lift-and-project cutting plane algorithm for mixed 0-1 programs. *Math. Program.* 58, 3 (1993), 295–324.
- [17] BALAS, E., AND ZEMEL, E. Facets of the knapsack polytope from minimal covers. *SIAM Journal on Applied Mathematics* 34 (1978), 119–148.
- [18] BARAHONA, F., AND ANBIL, R. The volume algorithm: Producing primal solutions with a subgradient method. *Mathematical Programming* 87 (2000), 385–399.
- [19] BARAHONA, F., AND LADÁNYI, L. Branch and cut based on the volume algorithm: Steiner trees in graphs and Max-Cut. *RAIRO Operations Research* 40, 1 (2006), 53–73.
- [20] BARAHONA, F., AND MAHJOUR, A. R. On the cut polytope. *Math. Program.* 36, 2 (1986), 157–173.
- [21] BATTITI, R. Reactive search: Toward self-tuning heuristics. In *Modern Heuristic Search Methods*, C. R. R. V. J. Rayward-Smith, I. H. Osman and G. D. Smith, Eds. John Wiley & Sons Ltd., Chichester, 1996, pp. 61–83.
- [22] BATTITI, R., AND PROTASI, M. Reactive search, a history-sensitive heuristic for MAX-SAT. *J. Exp. Algorithmics* 2 (1997), 2.
- [23] BATTITI, R., AND PROTASI, M. Approximate algorithms and heuristics for MAX-SAT. In *Handbook of Combinatorial Optimization*, D.-Z. Du and P. Pardalos, Eds. Kluwer Academic Publishers, 1998, pp. 77–148.
- [24] BATTITI, R., AND TECCHIOILLI, G. The reactive tabu search. *ORSA Journal on Computing* 6, 2 (1994), 126–140.
- [25] BAZGAN, C., SANTHA, M., AND TUZA, Z. On the approximation of finding a(nother) hamiltonian cycle in cubic hamiltonian graphs. *J. Algorithms* 31, 1 (1999), 249–268.
- [26] BEALE, E., AND TOMLIN, J. Special facilities in a general mathematical programming system for non-convex problems using ordered sets of variables. In *OR 69: Proceedings of the fifth international conference on operational research* (London, New-York, 1970), Tavistock Publications, pp. 447–454.
- [27] BEASLEY, J. Heuristic algorithms for the unconstrained binary quadratic programming problem. Tech. rep., Management School, Imperial College, London, UK, 1998.
- [28] BJÖRKLUND, A., AND HUSFELDT, T. Finding a path of superlogarithmic length. *SIAM Journal on Computing* 32, 6 (2003), 1395–1402.
- [29] BJÖRKLUND, A., HUSFELDT, T., AND KHANNA, S. Approximating longest directed paths and cycles. In *Automata, Languages and Programming 31st International Colloquium, ICALP 2004, Turku, Finland, July 12-16, 2004*. (2004), vol. 3142/2004 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, pp. 222–233.
- [30] BORCHERS, B., AND FURMAN, J. A two-phase exact algorithm for MAX-SAT and weighted MAX-SAT problems. *Journal of Combinatorial Optimization* 2, 4 (1999), 299–306.

- [31] BURER, S., MONTEIRO, R. D. C., AND ZHANG, Y. Rank-two relaxation heuristics for MAX-CUT and other binary quadratic programs. *SIAM J. on Optimization* 12, 2 (2002), 503–521.
- [32] CAPRARA, A., AND FISCHETTI, M. Branch-and-cut algorithms. In *Annotated Bibliographies in Combinatorial Optimization*. Wiley, New York, USA, 1997, pp. 45–64.
- [33] CHVÁTAL, V. Edmonds polytopes and a hierarchy of combinatorial problems. *Discrete Mathematics* 4 (1973), 304–337.
- [34] COOK, S. A. The complexity of theorem-proving procedures. In *STOC '71: Proceedings of the third annual ACM symposium on Theory of computing* (New York, NY, 1971), ACM, pp. 151–158.
- [35] COPPERSMITH, D., GAMARNIK, D., HAJIAGHAYI, M., AND SORKIN, G. B. Random MAX SAT, random MAX CUT, and their phase transitions. *Random Struct. Algorithms* 24, 4 (2004), 502–545.
- [36] CRAINIC, T. G., GENDREAU, M., HANSEN, P., AND MLADENović, N. Cooperative parallel variable neighborhood search for the p-median. *Journal of Heuristics* 10, 3 (2004), 293–314.
- [37] CRESZENZI, P., SILVESTRI, R., AND TREVISAN, L. To weight or not to weight: Where is the question? In *Fourth Israel Symposium on Theory of Computing and Systems, ISTCS 1996* (1996), pp. 68–77.
- [38] CROWDER, H., JOHNSON, E. L., AND PADBERG, M. Solving large-scale zero-one linear programming problems. *Operations Research* 31, 5 (1983), 803–834.
- [39] DAKIN, R. J. A tree-search algorithm for mixed integer programming problems. *The Computer Journal* 8, 3 (1965), 250–255.
- [40] DANTZIG, G., FULKERSON, R., , AND JOHNSON, S. M. Solution of a large-scale traveling salesman problem. *Operations Research* 2 (1954), 393–410.
- [41] DAVIS, M., LOGEMANN, G., AND LOVELAND, D. A machine program for theorem-proving. *Commun. ACM* 5, 7 (1962), 394–397.
- [42] DE BOER, P., KROESE, D., MANNOR, S., AND RUBINSTEIN, R. A tutorial on the cross-entropy method. *Annals of Operations Research* 134 (2005), 19–67.
- [43] DE FALCO, I., R., D. B., TARANTINO, E., AND VACCARO, R. Improving search by incorporating evolution principles in parallel tabu search. In *Proceedings of the IEEE World Congress on Computational Intelligence, 1994* (1994), vol. 2, pp. 823–828.
- [44] DE LA VEGA, W. F., AND KARPINSKI, M. On the approximation hardness of dense TSP and other path problems. *Inf. Process. Lett.* 70, 2 (1999), 53–55.
- [45] DE LA VEGA, W. F., AND KENYON-MATHIEU, C. Linear programming relaxations of max-cut. In *SODA '07: Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms* (Philadelphia, PA, USA, 2007), Society for Industrial and Applied Mathematics, pp. 53–61.
- [46] DELORME, C., AND POLJAK, S. Combinatorial properties and complexity of a max-cut approximation. *Eur. J. Comb.* 14, 4 (1993), 313–333.
- [47] DELORME, C., AND POLJAK, S. Laplacian eigenvalues and the maximum cut problem. *Math. Program.* 62, 3 (1993), 557–574.

- [48] DELORME, C., AND POLJAK, S. The performance of an eigenvalue bound on the max-cut problem in some classes of graphs. *Discrete Math.* 111, 1-3 (1993), 145–156.
- [49] DUARTE, A., ÁNGEL SÁNCHEZ, FERNÁNDEZ, F., AND CABIDO, R. A low-level hybridization between memetic algorithm and VNS for the MAX-CUT problem. In *GECCO '05: Proceedings of the 2005 conference on Genetic and evolutionary computation* (New York, NY, USA, 2005), ACM Press, pp. 999–1006.
- [50] DUMITRESCU, I., AND STÜTZLE, T. Combinations of local search and exact algorithms. In *Applications of Evolutionary Computing. Proceedings of EvoWorkshops 2003: EvoBIO, EvoCOP, EvoIASP, EvoMUSART, EvoROB, and EvoSTIM* (2003), vol. 2611, Springer Berlin / Heidelberg, pp. 211–223.
- [51] DUMITRESCU, I., AND STÜTZLE, T. A survey of methods that combine local search and exact algorithms. Tech. Rep. AIDA-03-07, FG Intellektik, FB Informatik, TU Darmstadt, Germany, 2003.
- [52] EVERETT, HUGH, I. Generalized lagrange multiplier method for solving problems of optimum allocation of resources. *OPERATIONS RESEARCH* 11, 3 (1963), 399–417.
- [53] FEIGE, U., AND GOEMANS, M. Approximating the value of two prover proof systems, with applications to MAX 2SAT and MAX DICUT. In *ISTCS '95: Proceedings of the 3rd Israel Symposium on the Theory of Computing Systems (ISTCS'95)* (Washington, DC, USA, 1995), IEEE Computer Society, pp. 182–189.
- [54] FEO, T. A., AND RESENDE, M. A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters* 8, 2 (1989), 67–71.
- [55] FEO, T. A., AND RESENDE, M. Greedy randomized adaptive search procedures. *Journal of Global Optimization* 6 (1995), 109–133.
- [56] FESTA, P., PARDALOS, P. M., RESENDE, M. G. C., AND RIBEIRO, C. G. GRASP and VNS for Max-Cut. In *MIC'2001: Proceedings of the Fourth Metaheuristics International Conference* (2001), pp. 371–376.
- [57] FESTA, P., PARDALOS, P. M., RESENDE, M. G. C., AND RIBEIRO, C. G. Randomized heuristics for the MAX-CUT problem. Tech. rep., AT&T Labs Research, 2002.
- [58] FIDUCCIA, C. M., AND MATTHEYSES, R. M. A linear-time heuristic for improving network partitions. In *DAC '82: Proceedings of the 19th conference on Design automation* (Piscataway, NJ, USA, 1982), IEEE Press, pp. 175–181.
- [59] FISHER, M. L. Optimal solution of scheduling problems using lagrange multipliers: Part i. *OPERATIONS RESEARCH* 21, 5 (1973), 1114–1127.
- [60] FISHER, M. L., AND SHAPIRO, J. F. Constructive duality in integer programming. *SIAM J. Appl. Math.* 27 (1974), 31–52.
- [61] FRENCH, A. P., AND WILSON, J. M. An LP-based heuristic procedure for the generalized assignment problem with special ordered sets. *Comput. Oper. Res.* 34, 8 (2007), 2359–2369.
- [62] FÜRER, M., AND RAGHAVACHARI, B. Approximating the minimum degree spanning tree to within one from the optimal degree. In *SODA '92: Proceedings of the third annual ACM-SIAM symposium on Discrete algorithms* (Philadelphia, PA, USA, 1992), Society for Industrial and Applied Mathematics, pp. 317–324.

- [63] GABOW, H. N. Finding paths and cycles of superpolylogarithmic length. In *STOC '04: Proceedings of the thirty-sixth annual ACM symposium on Theory of computing* (New York, NY, USA, 2004), ACM Press, pp. 407–416.
- [64] GARCÍA-LÓPEZ, F., MELIÁN-BATISTA, B., MORENO-PÉREZ, J. A., AND MORENO-VEGA, J. M. The parallel variable neighborhood search for the p-median problem. *Journal of Heuristics* 8, 3 (2002), 375–388.
- [65] GAREY, M. R., JOHNSON, D. S., AND STOCKMEYER, L. Some simplified NP-complete problems. In *STOC '74: Proceedings of the sixth annual ACM symposium on Theory of computing* (New York, NY, USA, 1974), ACM Press, pp. 47–63.
- [66] GARG, N., AND VAZIRANI, V. A polyhedron with all s-t cuts as vertices, and adjacency of cuts. In *In Proceedings of the 3rd Integer Programming and Combinatorial Optimization Conference* (1993), pp. 281–289.
- [67] GILMORE, P., AND GOMORY, R. A linear programming approach to the cutting- stock problem. *Operations Research* 9 (1961), 849–859.
- [68] GILMORE, P., AND GOMORY, R. A linear programming approach to the cutting stock problem—part ii. *Operations Research* 11 (1963), 863–888.
- [69] GLOVER, F. Convexity cuts and cut search. *Operations Research* 21, 1 (1973), 123–134.
- [70] GLOVER, F. Tabu search—part i. *ORSA Journal on Computing* 1, 3 (1989), 190–206.
- [71] GLOVER, F. Tabu thresholding: Improved search by nonmonotonic trajectories. *ORSA Journal on Computing* 7, 4 (1995), 426–442.
- [72] GLOVER, F. Tabu search and adaptive memory programming – Advances, applications and challenges. In *Advances in Metaheuristics, Optimization, and Stochastic Modeling Technologies*, R. Barr, R. Helgason, and J. L. Kennington, Eds., Interfaces in Operations Research and Computer Science. Kluwer, 1997, pp. 1–75.
- [73] GLOVER, F. Tabu search fundamentals and uses, June, 1995.
- [74] GLOVER, F., AND LAGUNA, M. Tabu search. In *Modern Heuristic Techniques for Combinatorial Problems*, C. Reeves, Ed. John Wiley & Sons, Inc., New York, NY, USA, 1993, pp. 70–150.
- [75] GLOVER, F., AND LAGUNA, M. General purpose heuristics for integer programming—part i. *Journal of Heuristics* 2 (1997), 343–358.
- [76] GLOVER, F., AND LAGUNA, M. *Tabu Search*. Kluwer Academic Publishers, Boston, USA, 1997.
- [77] GLOVER, F., AND LAGUNA, M. Fundamentals of scatter search and path relinking. *Control and Cybernetics* 29 (1999), 653–684.
- [78] GLOVER, F. W., AND KOCHENBERGER, G. A., Eds. *Handbook of Metaheuristics*, vol. 57 of *International Series in Operations Research & Management Science*. Springer, New York, 2003.
- [79] GOEMANS, M. X., AND WILLIAMSON, D. P. .879-approximation algorithms for MAX CUT and MAX 2SAT. In *STOC '94: Proceedings of the twenty-sixth annual ACM symposium on Theory of computing* (New York, NY, USA, 1994), ACM Press, pp. 422–431.

- [80] GOMORY, R. Solving linear programming problems in integers. In *Combinatorial Analysis, Proceedings of Symposia in Applied Mathematics 10* (Providence, RI, 1960), AMS, pp. 211–216.
- [81] GOMORY, R. E. Outline of an algorithm for integer solutions to linear programs. *Bulletin of the American Mathematical Society* 64, 5 (1958), 275–278.
- [82] GRÖTSCHEL, M., AND HOLLAND, O. Solution of large-scale symmetric travelling salesman problems. *Math. Program.* 51, 2 (1991), 141–202.
- [83] GU, Z., NEMHAUSER, G. L., AND SAVELSBERGH, M. W. P. Lifted cover inequalities for 0-1 integer programs: Computation. *INFORMS JOURNAL ON COMPUTING* 10, 4 (1998), 427–437.
- [84] GU, Z., NEMHAUSER, G. L., AND SAVELSBERGH, M. W. P. Lifted cover inequalities for 0-1 integer programs: Complexity. *INFORMS Journal on Computing* 11, 1 (1999), 117–123.
- [85] HADLOCK, F. Finding a maximum cut of a planar graph in polynomial time. *SIAM Journal of Computing* 4 (1975), 221–225.
- [86] HANSEN, P., AND JAUMARD, B. Algorithms for the maximum satisfiability problem. *Computing* 44, 4 (1990), 279–303.
- [87] HANSEN, P., JAUMARD, B., MLADENović, N., AND PARREIRA, A. Variable neighbourhood search for maximum weighted satisfiability problem. Tech. Rep. G-2000-62, Les Cahiers du GERAD, 2000.
- [88] HANSEN, P., MLADENović, N., AND PEREZ-BRITOS, D. Variable neighborhood decomposition search. *Journal of Heuristics* 7, 4 (2001), 335–350.
- [89] HÅSTAD, J. Some optimal inapproximability results. *J. ACM* 48, 4 (2001), 798–859.
- [90] HELD, M., AND KARP, R. M. The traveling salesman problem and minimum spanning trees. *Operations Research* 18 (1970), 1138–1162.
- [91] HELMBERG, C., AND RENDL, F. A spectral bundle method for semidefinite programming. *SIAM J. on Optimization* 10, 3 (1999), 673–696.
- [92] HIRSCH, E. A. A new algorithm for MAX-2-SAT. In *17th Annual Symposium on Theoretical Aspects of Computer Science, STACS 2000* (London, UK, 2000), G. Goos, J. Hartmanis, and J. van Leeuwen, Eds., Springer-Verlag, pp. 65–73.
- [93] HOFFMAN, K., AND PADBERG, M. Improving LP-representations of zero-one linear programs for branch-and-cut. *ORSA Journal on Computing* 3 (1991), 121–134.
- [94] HOMER, S., AND PEINADO, M. Design and performance of parallel and distributed approximation algorithms for Maxcut. *J. Parallel Distrib. Comput.* 46, 1 (1997), 48–61.
- [95] JAGOTA, A., AND SANCHIS, L. A. Some Experimental and Theoretical Results on Test Case Generators for the Maximum Clique Problem. Tech. Rep. 93-69, DIMACS, October 1993.
- [96] JÜNGER, M., REINELT, G., AND THIENEL, S. Practical problem solving with cutting plane algorithms in combinatorial optimization. In *Combinatorial Optimization*. American Mathematical Society, Providence, Rhode Island, USA, 995, pp. 111–152.
- [97] KARGER, D. R., MOTWANI, R., AND RAMKUMAR, G. D. S. On approximating the longest path in a graph. *Journal Algorithmica*, 1 (1997), 82–98.

- [98] KARP, R. Reducibility among combinatorial problems. In *COMPLEXITY OF COMPUTER COMPUTATIONS*, R. Miller and J. Thatcher, Eds. Plenum Press, New York, NY, USA, 1972.
- [99] KIM, S.-H., KIM, Y.-H., AND MOON, B.-R. A Hybrid Genetic Algorithm for the MAX CUT Problem. In *GECCO 2001: Proceedings of the Genetic Evolutionary Computation Conference* (2001), H. Beyer, E. Cantu-Paz, D. Goldberg, I. Parmee, L. Spector, and D. Whitley, Eds., Morgan Kaufmann Publishers, pp. 416–426.
- [100] KLOSE, A. An LP-based heuristic for two-stage capacitated facility location problems. *The Journal of the Operational Research Society* 50, 2 (1999), 157–166.
- [101] LAND, A. H., AND DOIG, A. G. An automatic method of solving discrete programming problems. *Econometrica* 28, 3 (1960), 497–520.
- [102] LEWIN, M., LIVNAT, D., AND ZWICK, U. Improved Rounding Techniques for the MAX 2-SAT and MAX DI-CUT Problems. In *Proceedings of the 9th International IPCO Conference on Integer Programming and Combinatorial Optimization* (London, UK, 2002), Springer-Verlag, pp. 67–82.
- [103] LORIE, J., AND SAVAGE, L. J. Three problems in capital rationing. *Journal of Business* (1955), 229–239.
- [104] LOVÁSZ, L., AND SCHRIJVER, A. Cones of matrices and setfunctions, and 0-1 optimization. *SIAM J. Opt.* 1, 2 (1991), 166–190.
- [105] LUENBERGER, D. G. *Linear and Nonlinear Programming*. Addison-Wesley, Reading, MA, USA, 1984.
- [106] MATUURA, S., AND MATSUI, T. 0.863-Approximation algorithm for MAX DICUT problem. In *Approximation, Randomization, and Combinatorial Optimization: Algorithms and Techniques: 4th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems, APPROX 2001 and 5th International Workshop on Randomization and Approximation Techniques in Computer Science, RANDOM 2001 Berkeley, CA, USA, August 18-20, 2001* (Berlin / Heidelberg, Germany, 2001), M. Goemans, K. Jansen, J. D. Rolim, and L. Trevisan, Eds., Springer-Verlag, pp. 138–146.
- [107] MATUURA, S., AND MATSUI, T. 0.935-Approximation randomized algorithm for MAX 2SAT and its derandomization. Tech. Rep. METR 2001-03, Department of Mathematical Engineering and Information Physics, The University of Tokyo, September 2001.
- [108] McLOUGHLIN, III, J. F., AND CEDENO, W. The enhanced evolutionary tabu search and its application to the quadratic assignment problem. In *GECCO '05: Proceedings of the 2005 conference on Genetic and evolutionary computation* (New York, 2005), ACM, pp. 975–982.
- [109] MILLS, P., AND TSANG, E. Guided local search for solving SAT and weighted MAX-SAT problems. *J. Autom. Reason.* 24, 1-2 (2000), 205–223.
- [110] MLADENović, N., AND HANSEN, P. Variable neighborhood search. *Comput. Oper. Res.* 24, 11 (1997), 1097–1100.
- [111] MONIEN, B. How to find long paths efficiently. *Annals of Discrete Mathematics*, 25 (1985), 239–254.
- [112] NEMHAUSER, G. L., AND WOLSEY, L. Integer programming. In *Optimization*, A. R. K. G.L. Nemhauser and M. Todd, Eds., vol. 1 of *Handbooks in Operations Research and Management Science*. Elsevier Science Publishers B.V., Amsterdam, The Netherlands, 1989, pp. 447–528.

- [113] NEMHAUSER, G. L., AND WOLSEY, L. A. *Integer and Combinatorial Optimization*. John Wiley & Sons, Inc., 1999.
- [114] NGOC, N. V., AND TUZA, Z. Linear-Time Algorithms for the Max Cut Problem. *Combinatorics, Probability & Computing* 2 (1993), 201–210.
- [115] NIEDERMEIER, R., AND ROSSMANITH, P. New upper bounds for maximum satisfiability. *Journal of Algorithms* 36, 1 (2000), 63–88.
- [116] ORLOVA, G., AND DORFMAN, Y. Finding the maximal cut in a graph. *Engineering Cybernetics* 10 (1972), 502–504.
- [117] PADBERG, M., AND RINALDI, G. A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. *SIAM Rev.* 33, 1 (1991), 60–100.
- [118] PADBERG, M. W. On the facial structure of set packing polyhedra. *Mathematical Programming* 1, 5 (1973), 199–215.
- [119] PADBERG, M. W. A note on zero-one programming. *Operations Research* 23 (1975), 833–837.
- [120] PAPADIMITRIOU, C. *Computational Complexity*. Addison Wesley Publishing Company, 1994.
- [121] PAPADIMITRIOU, C., AND YANNAKAKIS, M. Optimization, approximation, and complexity classes. In *STOC '88: Proceedings of the twentieth annual ACM symposium on Theory of computing* (New York, NY, USA, 1988), ACM Press, pp. 229–234.
- [122] PARDALOS, P. M., PITSOULIS, L. S., AND RESENDE, M. G. C. A parallel GRASP for MAX-SAT problems. In *PARA '96: Proceedings of the Third International Workshop on Applied Parallel Computing, Industrial Computation and Optimization* (London, UK, 1996), Springer-Verlag, pp. 575–585.
- [123] POHL, I. A method for finding Hamilton paths and Knight's tours. *Commun. ACM* 10, 7 (1967), 446–449.
- [124] POHL, I., AND STOCKMEYER, L. Pohl-Warnsdorf – Revisited. In *ISC (2004)*, M. H. Hamza, Ed., IASTED/ACTA Press, pp. 129–132.
- [125] POLJAK, S., AND RENDL, F. Node and edge relaxations of the Max-Cut problem. *Computing* 52, 2 (June 1994), 123–137.
- [126] POLJAK, S., AND TURZÍK, D. A polynomial time heuristic for certain subgraph optimization problems with guaranteed worst case bound. *Discrete Math.* 58 (1986), 99–104.
- [127] POLJAK, S., AND TUZA, Z. The expected relative error of the polyhedral approximation of the Max-Cut. *Operations Research Letters* 16, 4 (1994), 191–198.
- [128] POLJAK, S., AND TUZA, Z. Maximum cuts and largest bipartite subgraphs. In *Combinatorial Optimization: Papers from the DIMACS Special Year*, W. Cook, L. Lovász, and P. D. Seymour, Eds. American Mathematical Society, Providence, Rhode Island, USA, 1995, pp. 181–244.
- [129] PUCHINGER, J., AND RAIDL, G. R. Combining metaheuristics and exact algorithms in combinatorial optimization: A survey and classification. In *Artificial Intelligence and Knowledge Engineering Applications: A Bioinspired Approach*, vol. 3562 of *Lecture Notes in Computer Science*. Springer, Berlin / Heidelberg, 2005, pp. 41–53.

- [130] RESENDE, M. G., AND RIBEIRO, C. C. GRASP with Path-relinking: Recent Advances and Applications. In *Metaheuristics: Progress as Real Problem Solvers*, T. Ibaraki, K. Nonobe, and M. Yagiura, Eds., vol. 32 of *Operations Research/Computer Science Interfaces*. Kluwer Academic Publishers, 2005, pp. 29–64.
- [131] RESENDE, M. G. C., PITSOULIS, L. S., AND PARDALOS, P. M. Approximate solution of weighted MAX-SAT problems using GRASP. In *Satisfiability Problem: Theory and Applications*, vol. 35. American Mathematical Society, Providence, Rhode Island, USA, 1997, pp. 393–405.
- [132] ROY, T. V., AND WOLSEY, L. Solving mixed integer programming problems using automatic reformulation. *Operations Research* 35, 1 (1987), 45–57.
- [133] RUBINSTEIN, R. Y. Cross-entropy and rare events for maximal cut and partition problems. *ACM Transactions on Modelling and Computer Simulation* 12, 1 (2002), 27–53.
- [134] S.A., I. *ILOG OPL Studio 3.5 Language Manual*, 2001.
- [135] S.A., I. *ILOG CPLEX 10.0 User's Manual*, 2006.
- [136] SAHNI, S., AND GONZALEZ, T. P-complete approximation problems. *J. ACM* 23, 3 (1976), 555–565.
- [137] SCHRIJVER, A. On cutting planes. *Annals of Discrete Mathematics* 9 (1980), 291–296.
- [138] SCHRIJVER, A. *Theory of Linear and Integer Programming*. John Wiley & Sons, Inc., New York, NY, USA, 1986.
- [139] SCHRIJVER, A. *Combinatorial Optimization*. Springer-Verlag, Berlin, Heidelberg, 2003.
- [140] SELMAN, B., AND KAUTZ, H. A. An empirical study of greedy local search for satisfiability testing. In *Eleventh National Conference on Artificial Intelligence (AAAI-93)* (1993), pp. 46–51.
- [141] SELMAN, B., LEVESQUE, H., AND MITCHELL, D. A new method for solving hard satisfiability problems. In *Tenth National Conference on Artificial Intelligence, San Jose, CA, July 1992* (1992), AAAI Press/MIT Press, pp. 440–446.
- [142] SELMAN, B., MITCHELL, D., AND LEVESQUE, H. Generating hard satisfiability problems. *Artificial Intelligence* 81 (1996), 17–29.
- [143] SEYMOUR, P. On minors of non-binary matroids. *Combinatorica* 1 (1981), 387–394.
- [144] SHAPIRO, J. F. Generalized lagrange multipliers in integer programming. *Operations Research* 19, 1 (1971), 68–76.
- [145] SHEN, H., AND ZHANG, H. Study of lower bound functions for MAX-2-SAT. In *19th National Conference on Artificial Intelligence (AAAI)* (2004).
- [146] SHEN, H., AND ZHANG, H. Improving exact algorithms for MAX-2-SAT. *Annals of Mathematics and Artificial Intelligence* 44, 4 (2005), 419–436.
- [147] SHERALI, H. D., AND ADAMS, W. P. A hierarchy of relaxation between the continuous and convex hull representations. *SIAM J. Discret. Math.* 3, 3 (1990), 411–430.
- [148] SHERALI, H. D., AND ADAMS, W. P. A hierarchy of relaxation between the continuous and convex hull representations. *SIAM J. Discret. Math.* 3, 3 (1990), 411–430.

- [149] SMYTH, K., HOOS, H., AND STÜTZLE, T. Iterated robust tabu search for MAX-SAT. In *Advances in Artificial Intelligence: 16th Conference of the Canadian Society for Computational Studies of Intelligence, AI 2003, Halifax, Canada, June 11-13, 2003* (Berlin / Heidelberg, Germany, 2003), Y. X. B. Chaib-draa, Ed., Springer-Verlag, pp. 129–144.
- [150] SRIDHAR, V., PARK, J. S., AND GAVISH, B. LP-based heuristic algorithms for interconnecting token rings via source routing bridges. *Journal of Heuristics* 6, 1 (2000), 149–166.
- [151] TAILLARD, E. Robust taboo search for the quadratic assignment problem. *Parallel Computing* 17, 4–5 (1991), 443–455.
- [152] TUTTE, W. T. On hamiltonian circuits. *Journal of the London Mathematical Society* s1-21 (1946), 98–101.
- [153] TUY, H. Concave programming under linear constraints. *Doklady Akademii Nauk SSSR* 158 (1964), 32–35.
- [154] UMETANI, S., YAGIURA, M., AND IBARAKI, T. An LP-based local search for one dimensional cutting stock problem. In *Proceedings of the 5th Metaheuristics International Conference (MIC2003)* (2003), pp. 75/1–8.
- [155] VASQUEZ, M., AND HAO, J.-K. A hybrid approach for the 0–1 multidimensional knapsack problem. In *Proceedings of the International Joint Conference on Artificial Intelligence 2001, IJCAI* (2001), pp. 328–333.
- [156] VISHWANATHAN, S. An approximation algorithm for finding long paths in hamiltonian graphs. *Journal of Algorithms* 50 (2004), 246–256.
- [157] VOUDOURIS, C. *Guided Local Search for Combinatorial Optimisation Problems*. PhD thesis, Department of Computer Science, University of Essex, 1997.
- [158] WAH, B. W., AND SHANG, Y. Discrete lagrangian-based search for solving MAX-SAT problems. In *IJCAI 1: The 15th International Joint Conference on Artificial Intelligence* (San Francisco, CA, USA, 1997), Morgan Kaufmann Publishers, pp. 378–383.
- [159] WALLACE, R. J., AND FREUDER, E. C. Comparative studies of constraint satisfaction and davis-putnam, algorithms for maximum satisfiability problems. In *Cliques, Coloring, and Satisfiability*. American Mathematical Society, Providence, Rhode Island, USA, 1996, pp. 587–615.
- [160] WOLSEY, L. A. Faces for a linear inequality in 01 variables. *Mathematical Programming* 8 (1975), 165–178.
- [161] WOLSEY, L. A. *Integer Programming*. John Wiley & Sons, New York, NY, USA, 1998.
- [162] YAGIURA, M., AND IBARAKI, T. Efficient 2 and 3-flip neighborhoods search algorithms for the MAX SAT. In *Computing and Combinatorics: 4th Annual International Conference, COCOON'98, Taipei, Taiwan, R.o.C., August 1998* (Berlin / Heidelberg, Germany, 1998), W.-L. Hsu and M.-Y. Kao, Eds., Lecture Notes in Computer Science, Springer-Verlag, pp. 105–116.
- [163] YANNAKAKIS, M. On the approximation of maximum satisfiability. In *SODA '92: Proceedings of the third annual ACM-SIAM symposium on Discrete algorithms* (Philadelphia, PA, USA, 1992), Society for Industrial and Applied Mathematics, pp. 1–9.
- [164] YOUNG, R. D. Hypercylindrically deduced cuts in zero-one integer programs. *Operations Research* 19, 6 (1971), 1393–1405.

- [165] ZEMEL, E. Lifting the facets of zero-one polytopes. *Mathematical Programming* 15 (1978), 251–261.
- [166] ZWICK, U. Analyzing the MAX 2-SAT and MAX DI-CUT approximation algorithms of Feige and Goemans. available from the <http://www.cs.tau.ac.il/~zwick/>, 2000.

Appendix A

Types of Graphs Generated by rudy

`rudy`¹ [91] is a public domain and machine independent random graph generator written by Rinaldi. `rudy` generates various types of graphs. We list some of these types that were used in our experimental study.

simple-random:

On a given size of graph (a number of nodes n and a density ρ) the arcs are uniformly and independently distributed (constructed) at random. The number of edges is equal to $\lfloor \frac{\rho n(n-1)}{200} \rfloor$. Recall that the number of edges of a complete graphs is $\frac{n(n-1)}{2}$.

toroidal grid:

A toroidal bidimensional grid with a given size (a height h and a width w). The number of nodes is equal to hw and the number of arcs is equal to $2hw$.

planar:

Generates a random planar graph of a given number of nodes n and *density* σ . The parameter σ is any real in the interval $[0,100]$. The number of edges of the graph is given by $\lfloor 3(n-2)\frac{\sigma}{100} \rfloor$. Recall that the maximum number of edges of a planar graph is $3(n-2)$.

“almost” planar:

The edge set constructed as union of two “almost maximal” planar graphs, with number of edges equal to 99 p.c. of the possible maximal number of edges, on the same set of nodes.

simplex:

Graph node is associated to a given d dimensional nonnegative integer vector, whose sum of the components is limited by some given positive integer number k ; two nodes are adjacent if the Euclidean distance of the two corresponding vectors is $\sqrt{2}$.

¹<http://www-user.tu-chemnitz.de/helmberg/semidef.html>

Appendix B

Some Details of Computational Results

B.1. Max-Di-Cut: 100 individual runs of TV

In the experimental study of our technique for the MAX-DI-CUT problem with source and sink we carried out 100 individual runs of the TV procedure to all instances of the eleventh class with the setting of the solution values found by `LPcut` set as the target values. The produced distributions of the iteration-to-target-value for each of the 54 instances of the eleventh class are shown in disjoint plots of Figures B.1 to B.5.

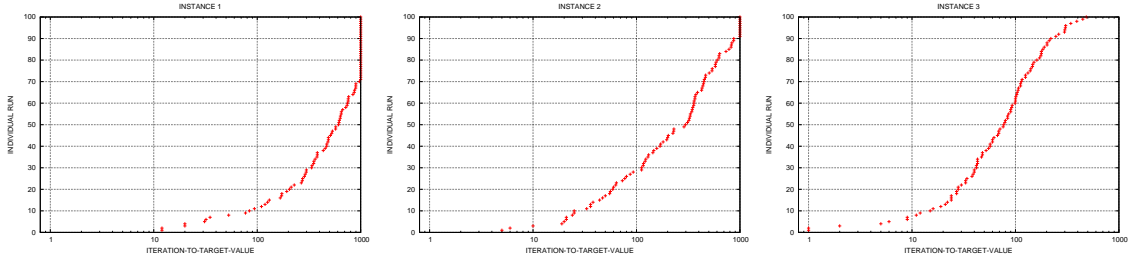


Figure B.1: The distribution of the iteration-to-target-value by 100 individual runs of the TV to the instances with ID numbers of 1 to 3 of the eleventh class for the MAX-DI-CUT.

B.2. Max-2-Sat: Computational Results for the 2nd class

Table B.1 shows the comparison of our technique and reference technique to the instances of the second test class for the MAX-2-SAT problem.

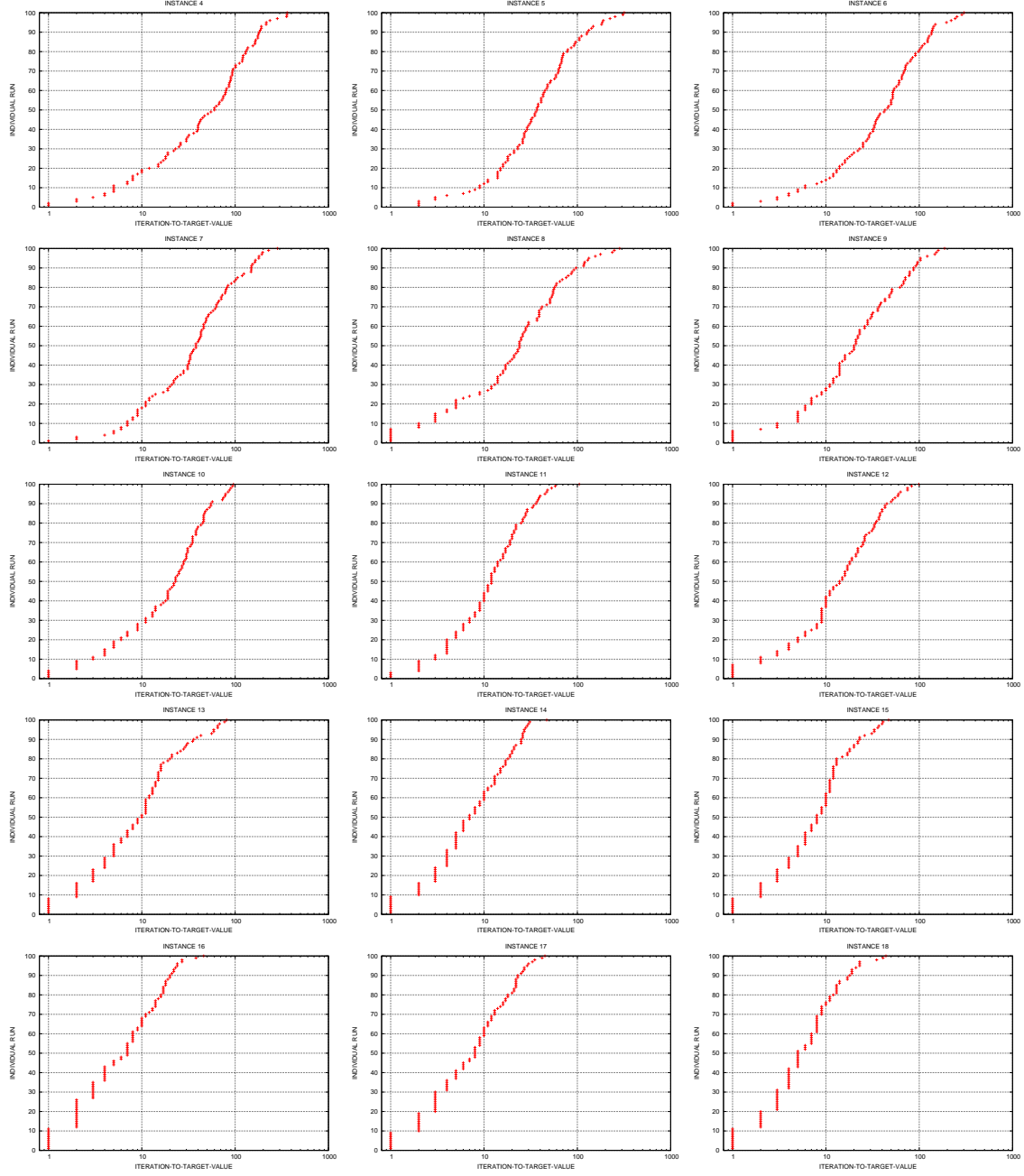


Figure B.2: The distribution of the iteration-to-target-value by 100 individual runs of the TV to the instances with ID numbers from 4 to 18 of the eleventh test class for the MAX-DI-CUT.

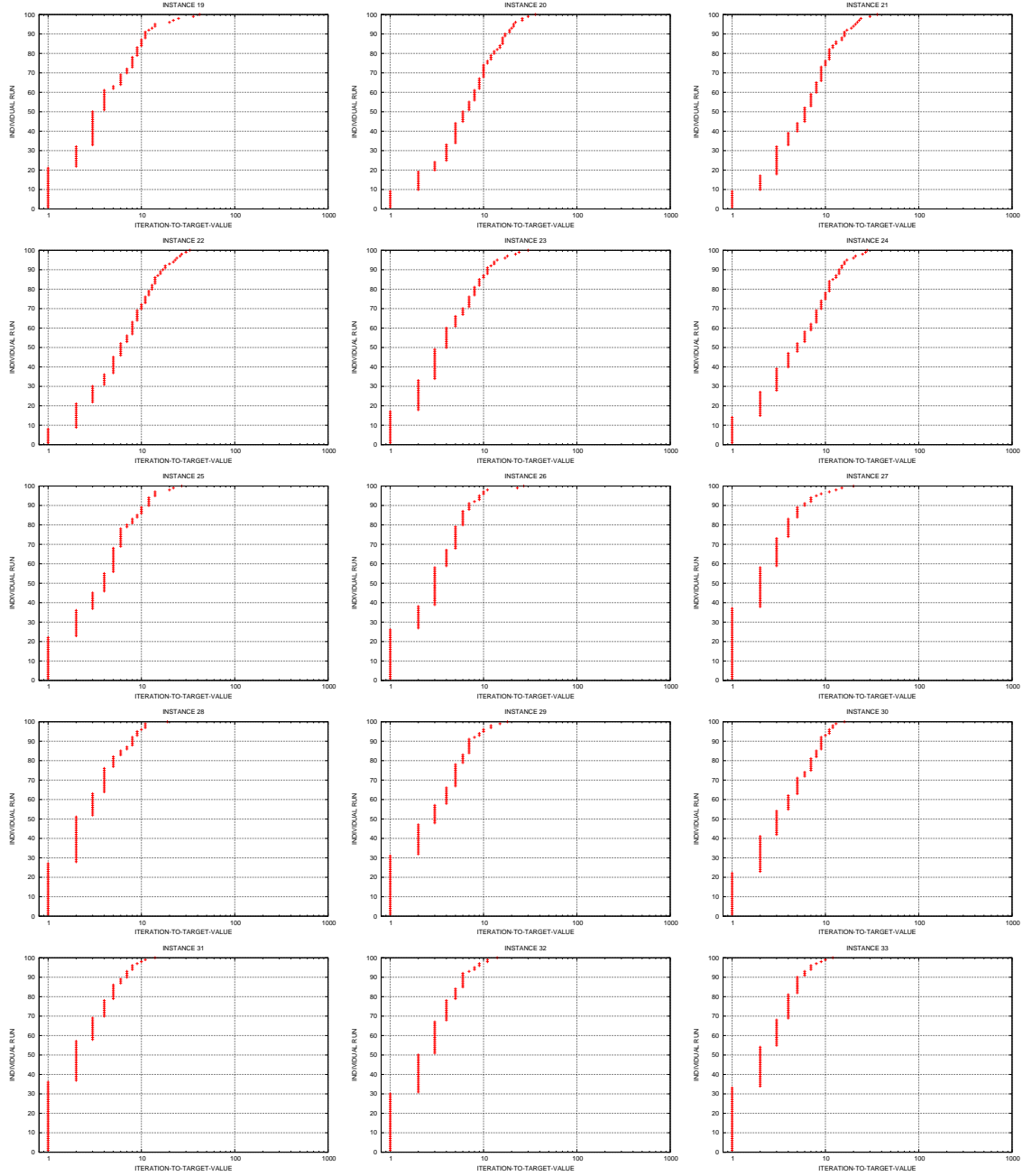


Figure B.3: The distribution of the iteration-to-target-value for 100 individual runs of the TV to the instances with ID numbers of 19 to 33 of the eleventh test class for the MAX-DI-CUT.

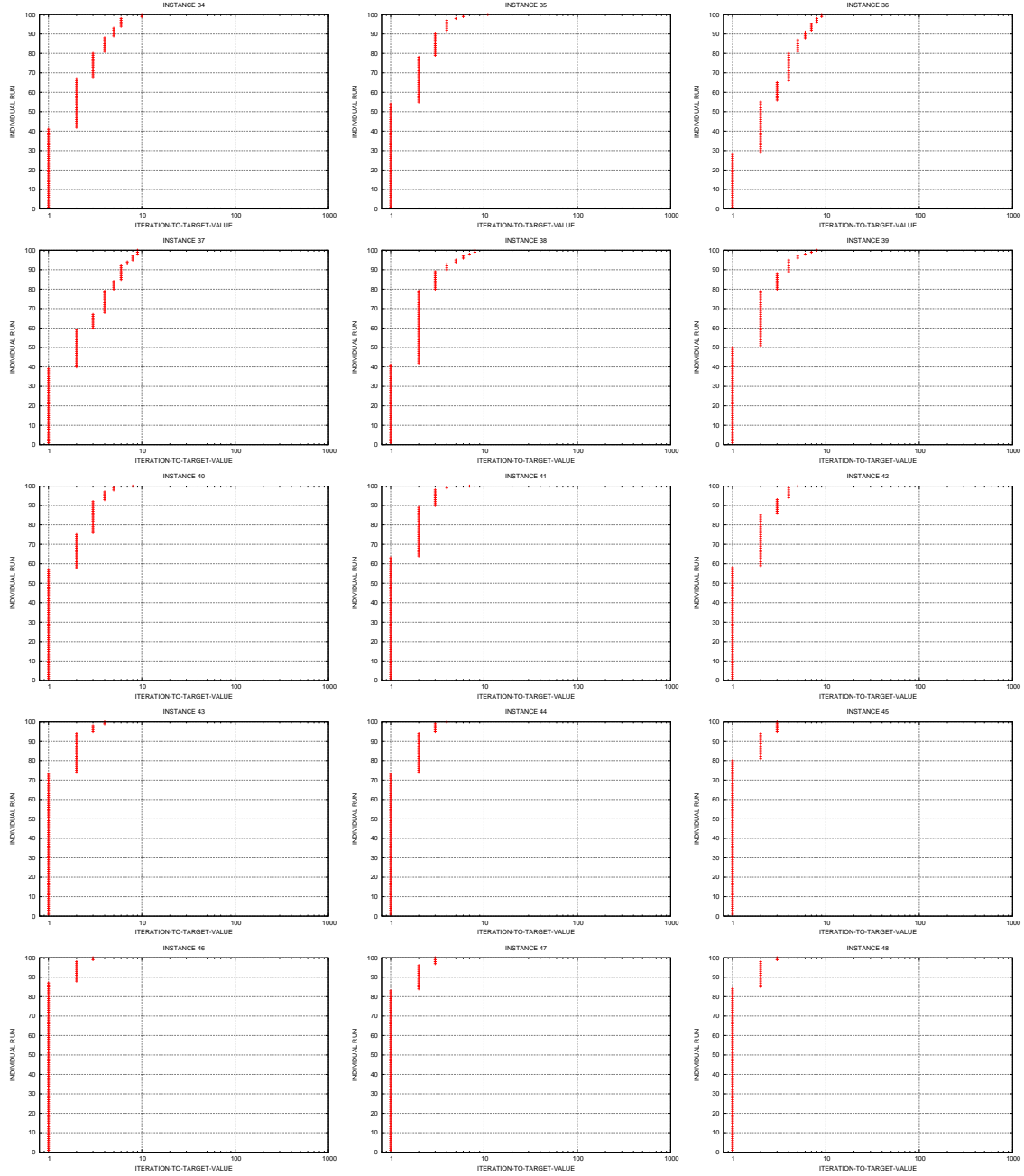


Figure B.4: The distribution of the iteration-to-target-value by 100 individual runs of the TV to the instances with ID numbers of 34 to 48 of the eleventh test class for the MAX-DI-CUT.

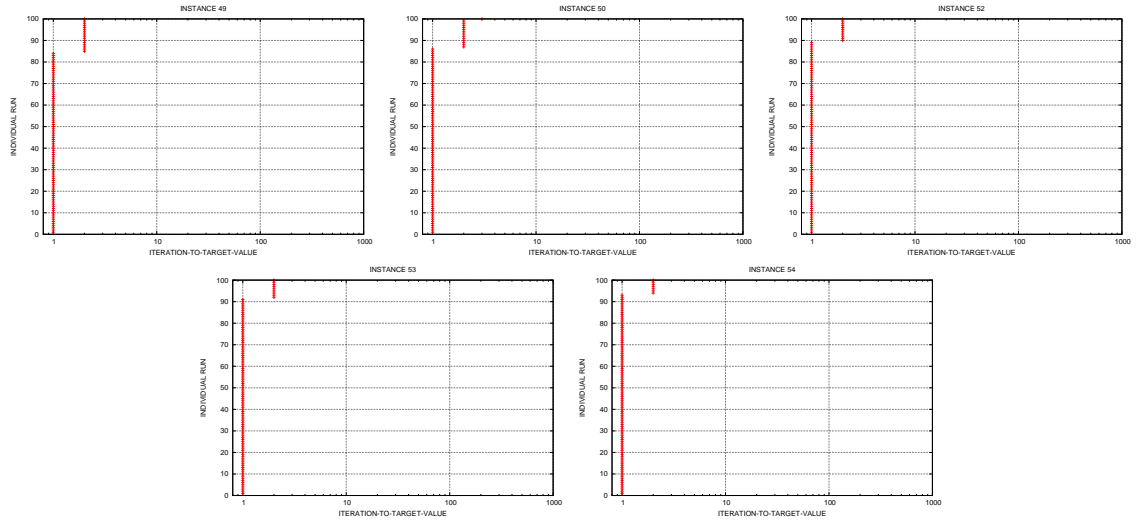


Figure B.5: The distribution of the iteration-to-target-value by 100 individual runs of the TV to the instances with ID numbers of 49 to 54 of the eleventh test class of MAX-DI-CUT.

Instance graph $G = (V, E)$			LPcut		REF		$\frac{LPcut}{REF} * 100$ p.c.	
ID	$ V $	density	obj. value	CPU time (sec)	obj. value	CPU time (sec)	ratio of obj. values	ratio of CPU times
1	800	6	15352	0,49	15393	151,1	99,73	0,32
2	800		15399	0,46	15425	157,2	99,83	0,29
3	800		15360	0,47	15393	151,2	99,79	0,31
4	800		15370	0,48	15427	158,7	99,63	0,30
5	800		15384	0,47	15429	149,5	99,71	0,31
6	800		7779018	0,66	7794527	161,7	99,80	0,41
7	800		7813647	0,73	7819829	160,5	99,92	0,45
8	800		7734481	0,63	7748670	161,6	99,82	0,39
9	800		7833307	0,63	7843257	161,2	99,87	0,39
10	800		7859423	0,68	7862669	160,6	99,96	0,42
11	800	0,5	787	0,06	793	72,4	99,24	0,08
12	800		766	0,07	777	71,8	98,58	0,10
13	800		778	0,07	797	71,1	97,62	0,10
14	800	1,47	4375	0,11	4388	84,9	99,70	0,13
15	800		4369	0,12	4381	78,2	99,73	0,15
16	800		4381	0,13	4386	81,7	99,89	0,16
17	800		4382	0,13	4391	79,8	99,80	0,16
18	800		24421	0,17	24451	97,5	99,88	0,17
19	800		24408	0,16	24455	93,8	99,81	0,17
20	800		23999	0,13	24037	98,4	99,84	0,13
21	800		24273	0,15	24326	96,4	99,78	0,16
22	2000	1	16672	1,742	16683	1847,4	99,93	0,09
23	2000		16654	1,862	16674	1826,0	99,88	0,10
24	2000		16602	2,132	16647	1525,2	99,73	0,14
25	2000		16639	1,852	16674	1755,4	99,79	0,11
26	2000		16624	1,832	16657	1708,8	99,80	0,11
27	2000		93128	3,855	93142	2152,1	99,98	0,18
28	2000		93119	2,673	93271	1950,3	99,84	0,14
29	2000		92415	2,182	92622	2097,4	99,78	0,10
30	2000		92229	2,923	92300	2005,9	99,92	0,15
31	2000		92452	2,192	92624	2129,5	99,81	0,10
32	2000	0,2	20475	1,512	21245	1472,8	96,38	0,10
33	2000		20798	1,973	21235	1380,9	97,94	0,14
34	2000		21032	2,163	21382	1210,3	98,36	0,18
35	2000	0,59	11052	1,341	11072	1076,6	99,82	0,12
36	2000		10999	1,102	11013	964,8	99,87	0,11
37	2000		11048	1,091	11046	1082,8	100,02	0,10
38	2000		11042	1,342	11046	944,5	99,96	0,14
39	2000		55470330	1,692	55515861	927,6	99,92	0,18
40	2000		54955443	1,511	55046257	913,1	99,84	0,17
41	2000		55379720	2,002	55404126	923,6	99,96	0,22
42	2000		55754236	1,562	55817844	887,6	99,89	0,18
43	1000	2	8307	0,3	8328	242,4	99,75	0,12
44	1000		8308	0,33	8320	246,7	99,86	0,13
45	1000		8299	0,3	8309	276,1	99,88	0,11
46	1000		8285	0,31	8318	245,7	99,60	0,13
47	1000		8325	0,32	8340	237,4	99,82	0,13
48	3000	0,13	5508	5,618	6000	4426,0	91,80	0,13
49	3000		5612	7,26	6000	4196,1	93,53	0,17
50	3000		5485	3,825	5927	4405,0	92,54	0,09
51	1000	1,18	5559	0,22	5564	142,2	99,91	0,15
52	1000		5538	0,19	5548	135,1	99,82	0,14
53	1000		5504	0,18	5511	144,3	99,87	0,12
54	1000		5545	0,18	5550	164,0	99,91	0,11

Table B.1: The computational results of the two algorithms to the MAX-2-SAT instances of the second class (Section 5.3.2): the ratio (p.c.) of the solution values found by LPcut and REF; the ratio of the CPU times needed by LPcut and REF.

B.3. Max-2-Sat: 60 individual runs of the TV

In the experimental study of our technique for the MAX-2-SAT problem we carried out 60 individual runs of the TV procedure to the each of 50 instances of the fourth test class. In this test the the solutions found by our technique were set as target solutions. This test produced the distributions of the iteration-to-target-value for the reference technique as shown in Figures B.6–B.9.

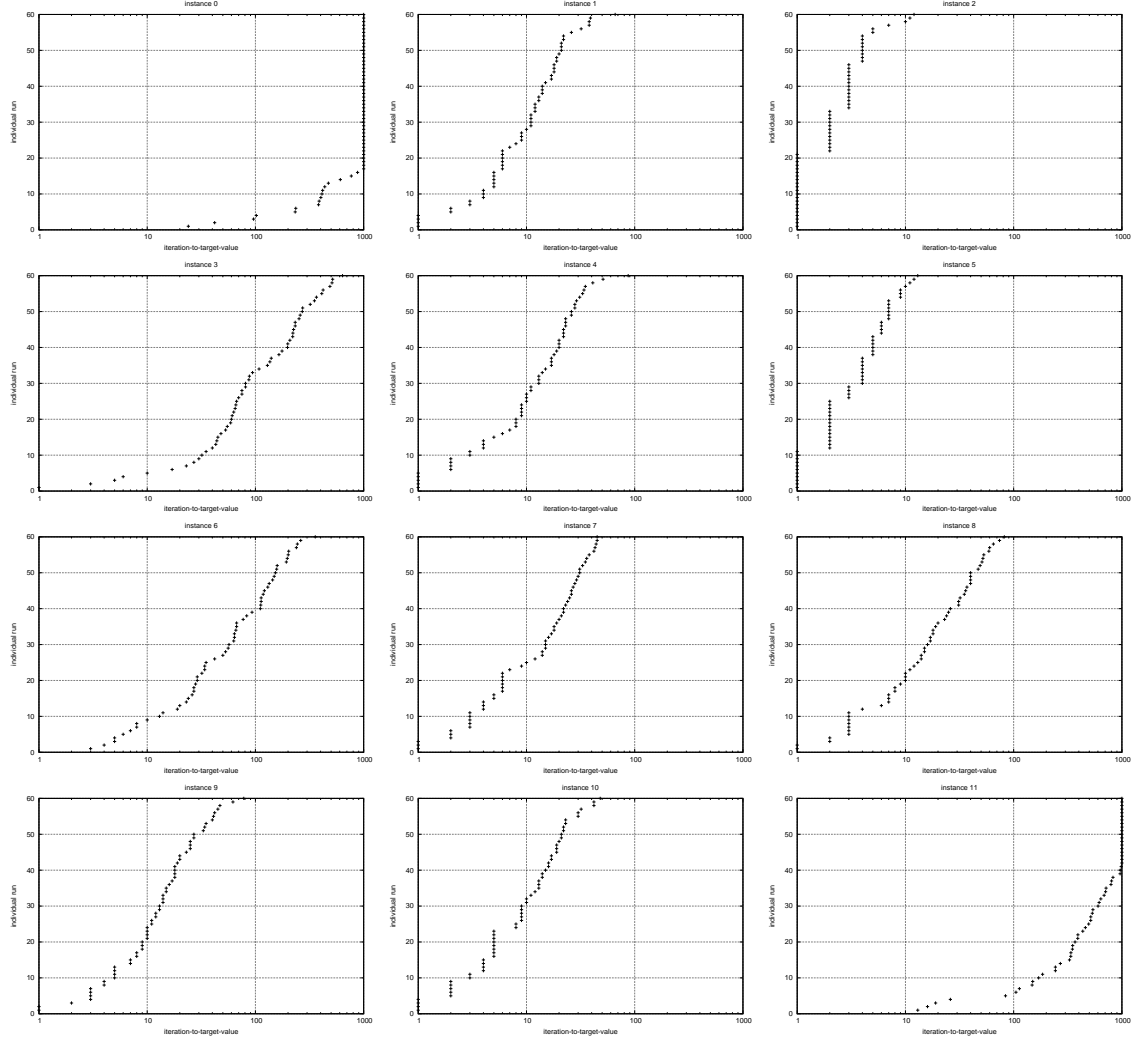


Figure B.6: The distribution of the iteration-to-target-value by 60 runs of the TV to the instances with ID numbers of 0 to 11 of the fourth class of MAX-2-SAT instances.

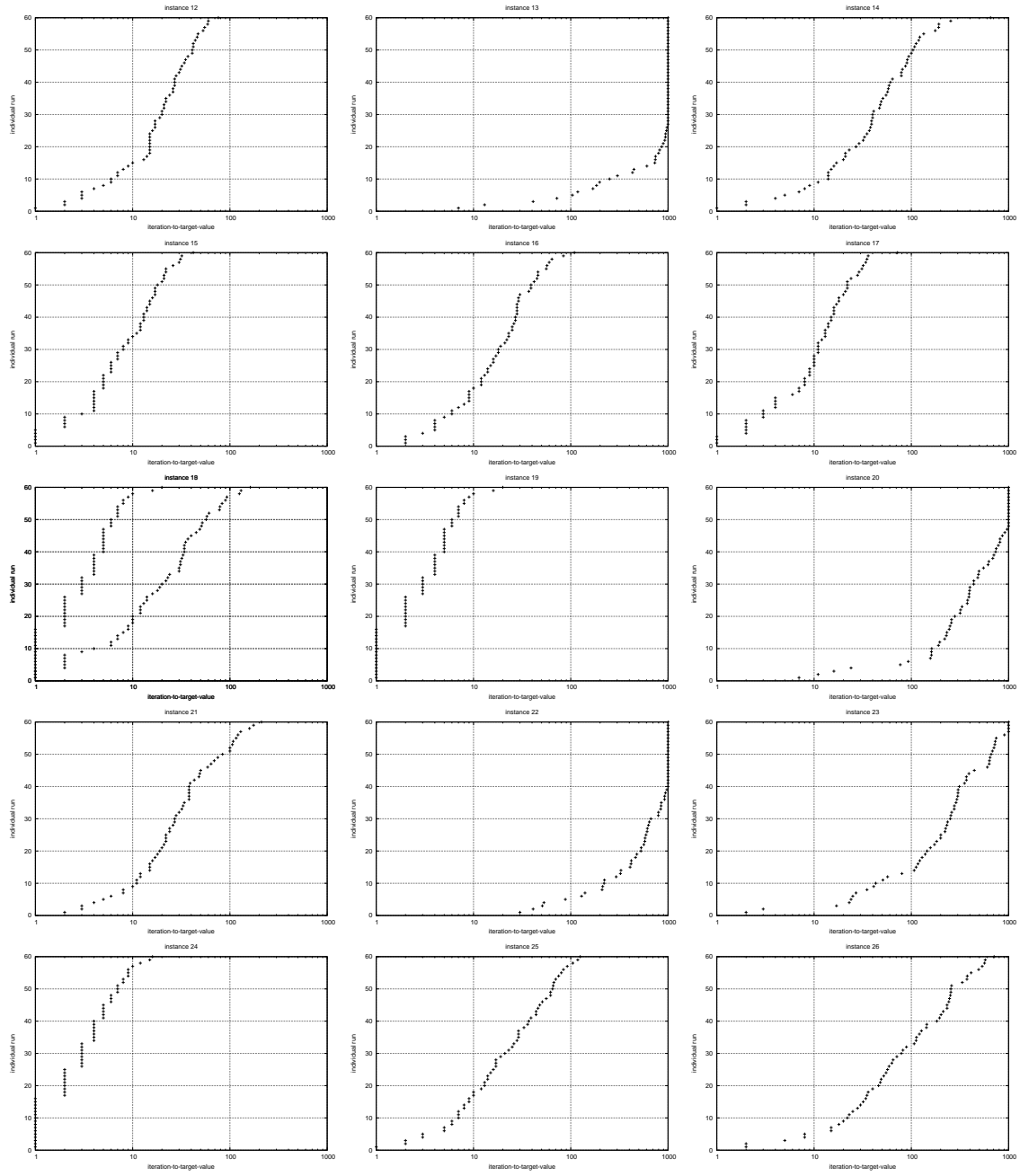


Figure B.7: The distribution of the iteration-to-target-value by 60 runs of the TV to the instances with ID numbers of 12 to 26 of the fourth class of MAX-2-SAT instances.

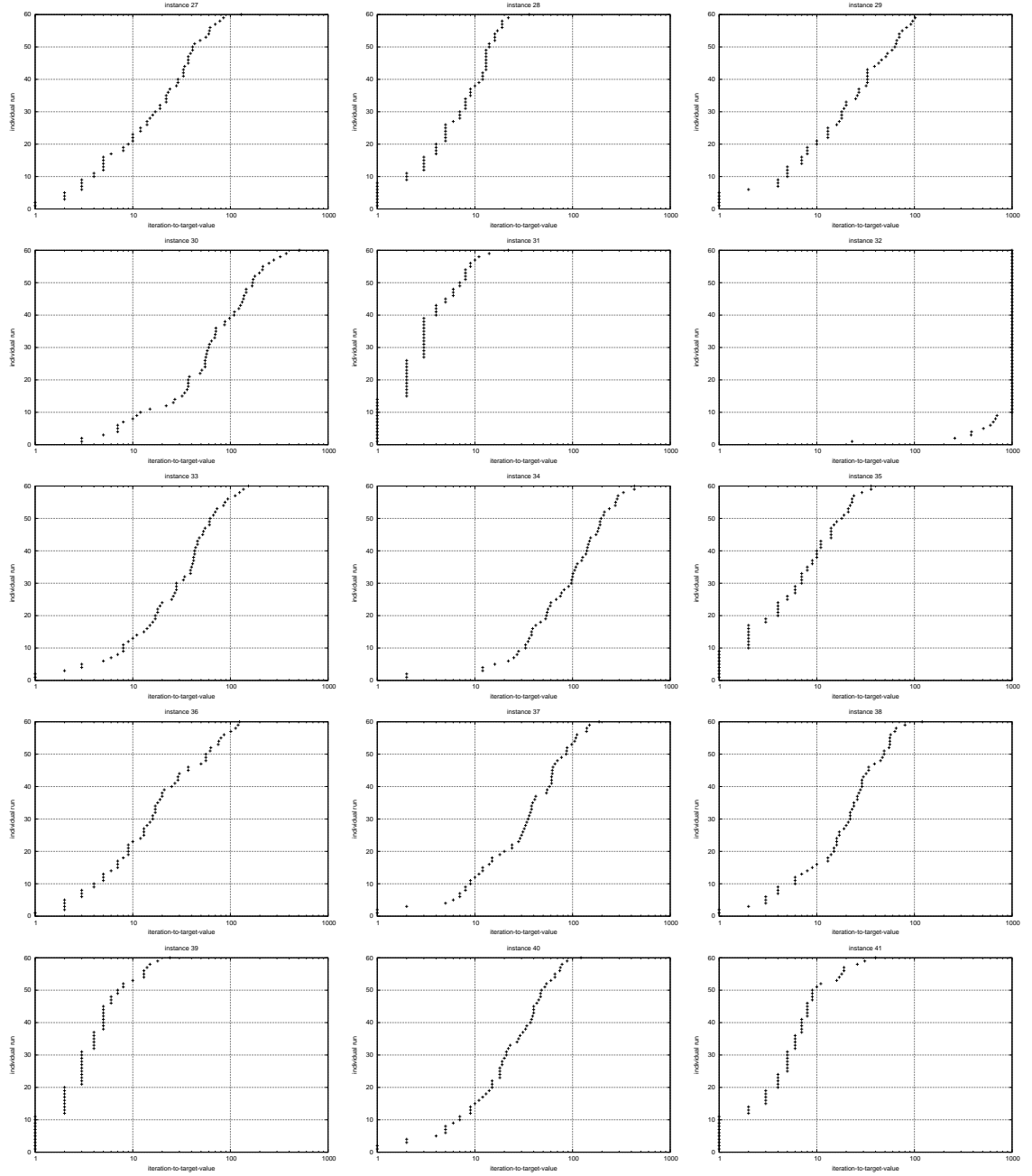


Figure B.8: The distribution of the iteration-to-target-value by 60 runs of the TV to the instances with ID numbers of 27 to 41 of the fourth class of MAX-2-SAT instances.

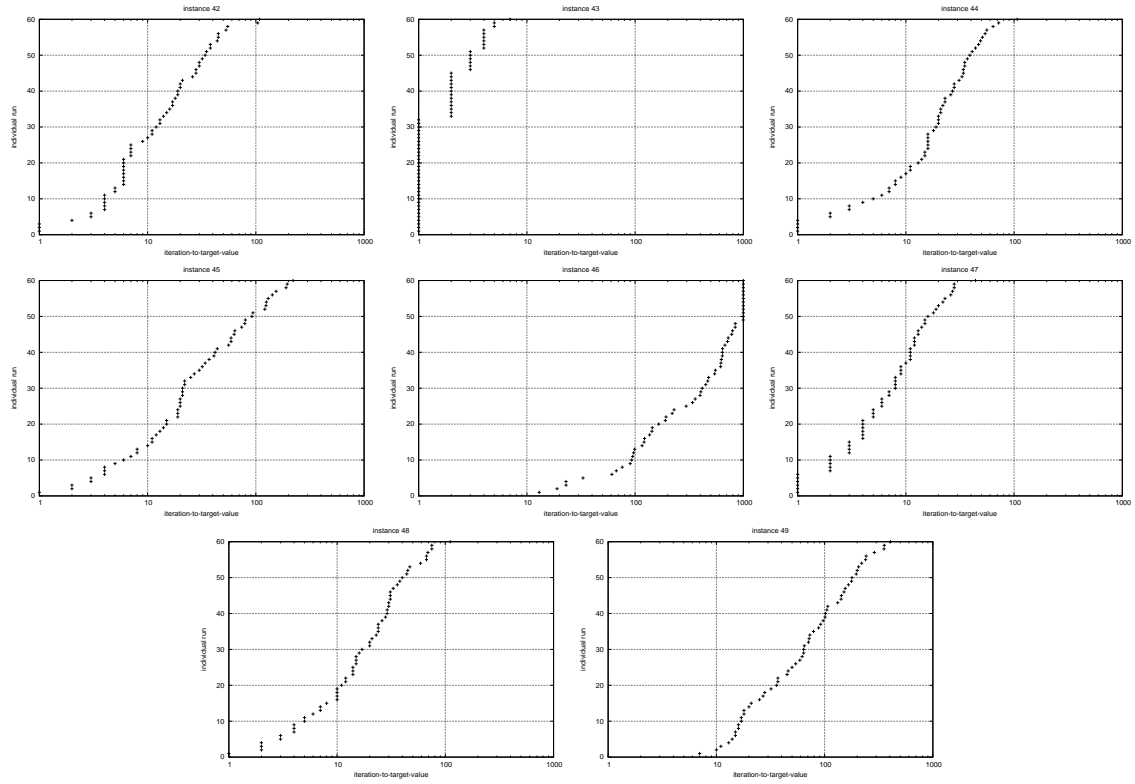


Figure B.9: The distribution of the iteration-to-target-value by 60 runs of the TV to the instances with ID numbers of 42 to 49 of the fourth class of MAX-2-SAT instances.

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit mit dem Titel “A Novel LP-based Local Search Technique – Fast and Quite Good” selbstständig und ohne Hilfe Dritter angefertigt habe. Sämtliche Hilfsmittel, Hilfen sowie Literaturquellen sind als solche kenntlich gemacht. Ausserdem erkläre ich hiermit, dass ich mich nicht anderweitig um einen entsprechenden Doktorgrad beworben habe.

Alaubek Avdil
Darmstadt, 04 Juni 2009

Wissenschaftlicher Werdegang

Alaubek Avdil
geboren am 23.01.1978.

Schulbildung

1985 – 1990: Allgemeine Grundschulabschluss der 5. Klasse
Schule Nr. 81, Ulaanbaatar, Mongolei

1990 – 1994: Allgemeine Hochschulreife
Schule Nr. 11, Ulaanbaatar, Mongolei

Hochschulausbildung

1994 – 1998: Bachelor of Science
Studium der Angewandten Mathematik
an der Mongolische Nationale Universität, Ulaanbaatar, Mongolei

1998 – 1999: Master of Science
Masterstudium der Mathematik
an der Mongolische Nationale Universität, Ulaanbaatar, Mongolei.